FACULTY OF SCIENCES
DEPARTMENT OF APPLIED MATHEMATICS, COMPUTER SCIENCE AND STATISTICS

Robustness of Classifiers to Adversarial Perturbations

Jonathan Peck

supervised by
Prof. Dr. Yvan SAEYS
Prof. Dr. Ir. Bart GOOSSENS
M.Sc. Joris ROELS

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Mathematical Informatics.

May 2017

Acknowledgements

First and foremost I would like to thank Yvan Saeys for providing me with the opportunity to do my thesis with him on this subject. The topic of adversarial perturbations is one which greatly intrigued me from the moment I read about it in the original paper of Szegedy et al. 2013, and I very much enjoyed studying it in greater depth.

Secondly, I wish to thank all my supervisors, Yvan Saeys, Bart Goossens and Joris Roels, for their support and helpful comments. They also provided me with access to the DAMBI computing cluster and GPU-accelerated machines at TELIN without which many of the results here would not have been feasible to compute. On that note, I also thank the entire DAMBI group for the positive feedback they gave me when I presented my thesis to them.

Finally, I thank my girlfriend Orla for her patience with me. Her support and understanding have helped me greatly during this past year, especially during frustrating and stressful times.

Jonathan Peck, May 2017

Permission for use of content

The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use.

In the case of any other use, the limitations of the copyright have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation.

Jonathan Peck, May 2017

Summary

The input-output mappings learned by state-of-the-art neural networks are significantly discontinuous. It is possible to cause a neural network used for image recognition to misclassify its input by applying very specific, hardly perceptible perturbations to the input, called *adversarial perturbations*. Many hypotheses have been proposed to explain the existence of these peculiar samples as well as several methods to mitigate them, but a proven explanation remains elusive. In this work, we take steps towards a formal characterization of adversarial perturbations by deriving lower bounds on the magnitudes of perturbations necessary to change the classification of neural networks. The proposed bounds can be computed efficiently, requiring time at most linear in the number of parameters and hyperparameters of the model for any given sample. This makes them suitable for use in model selection, when one wishes to find out which of several proposed classifiers is most robust to adversarial perturbations. They may also be used as a basis for developing techniques to increase the robustness of classifiers, since they enjoy the theoretical guarantee that no adversarial perturbation could possibly be any smaller than the quantities provided by the bounds. We experimentally verify the bounds on several data sets and neural network architectures, including Multi-Layer Perceptrons trained on Iris, Wine and Spambase as well as Convolutional Neural Networks trained on the MNIST and CIFAR-10 data sets. We find no violations in any of our tests. Additionally, the theoretical analysis suggests that the robustness of a neural network may significantly increase if one manages to slightly increase the robustness of its individual layers. Moreover, this effect appears to scale exponentially with the number of layers. We also construct variants of the Convolutional Neural Networks used in our tests which include Local Response Normalization layers at various points, and find that the robustness increases as one adds more normalization layers to the model while the accuracy remains about the same. Thus, the addition of normalization layers appears to be a viable strategy for increasing robustness.

Robustness of Classifiers to Adversarial Perturbations

Jonathan Peck

May 2017

Abstract

The input-output mappings learned by state-of-the-art neural networks are significantly discontinuous. It is possible to cause a neural network used for image recognition to misclassify its input by applying very specific, hardly perceptible perturbations to the input, called *adversarial perturbations*. Many hypotheses have been proposed to explain the existence of these peculiar samples as well as several methods to mitigate them, but a proven explanation remains elusive. In this work, we take steps towards a formal characterization of adversarial perturbations by deriving lower bounds on the magnitudes of perturbations necessary to change the classification of neural networks. The proposed bounds can be computed efficiently, requiring time at most linear in the number of parameters and hyperparameters of the model for any given sample. We experimentally verify the bounds on the MNIST and CIFAR-10 data sets and find no violations. Additionally, the theoretical analysis suggests that the robustness of a neural network may significantly increase if one manages to slightly increase the robustness of its individual layers. Moreover, this effect appears to scale up exponentially with the number of layers. We construct variants of the Convolutional Neural Networks used in our tests which include Local Response Normalization layers at various points, and find that the robustness increases as one adds more normalization layers to the model while the accuracy remains approximately the same. The addition of normalization layers thus seems to be a viable technique for increasing robustness without sacrificing too much accuracy.

Keywords: machine learning, adversarial perturbations, neural networks, robustness

1 Introduction

Despite their big successes in various AI tasks, neural networks are basically black boxes: there is no clear fundamental explanation how they are able to outperform the more classical approaches. This has led to the identification of several unexpected and counter-intuitive properties of neural networks. In particular, Szegedy et al. 2013 discovered that the input-output mappings learned by state-of-the-art neural networks are significantly discontinuous. It is possible to cause a neural network used for image recognition to misclassify its input by applying a very specific, hardly perceptible perturbation to the input. Szegedy et al. 2013 call these perturbations *adversarial perturbations*, and the inputs resulting from applying them to natural samples are called *adversarial examples*.

In this work, we hope to shed more light on the nature and cause of adversarial examples by deriving lower bounds on the magnitudes of perturbations necessary to change the classification of neural network classifiers. Such lower bounds are indispensable for developing rigorous methods that increase the robustness of classifiers without sacrificing accuracy. Since the bounds enjoy the theoretical guarantee that no adversarial perturbation could ever be any smaller, a method which increases these lower bounds potentially makes the classifier more robust. They may also aid model selection: if the bounds can be computed efficiently, then one can use them to compare different models with respect to their robustness to adversarial perturbations and select the model that scores the highest in this regard without the need for extensive empirical tests.

2 Related work

Goodfellow et al. 2015 introduce their widely accepted *linearity hypothesis*, which conjectures that the existence of adversarial perturbations is due to highly

linear behavior in modern neural networks. Since these models are so deep, linear behavior can cause small perturbations in the input to "snowball" into high perturbations of the output. Goodfellow et al. 2015 also discovered that adversarial examples generalize across different models. A specific adversarial example that was designed to fool a certain model A will often also fool a different model B. This happens with high probability even if both models have different architectures or are trained on different data sets.

Little formal investigation has been done into the nature of adversarial perturbations, however. Hence, it may still be possible to generate adversarial examples for classifiers using techniques which defy the proposed hypotheses. As such, there is a need to formally characterize the cause of adversarial examples. Fawzi et al. 2016 take a step in this direction by deriving precise bounds on the norms of adversarial perturbations of arbitrary classifiers in terms of the curvature of the decision boundary. Their analysis encourages to impose geometric constraints on this curvature in order to improve robustness. However, it is not obvious how such constraints relate to the parameters of the models and hence how one would best implement such constraints in practice. In this work, we derive lower bounds on the robustness of neural networks directly in terms of their model parameters. We consider only feedforward networks comprised of convolutional layers, pooling layers, fully-connected layers, softmax layers, Batch Normalization layers and Local Response Normalization layers.

3 Theoretical framework

In the following, $\|\cdot\|$ denotes the Euclidean norm and $\|\cdot\|_F$ denotes the Frobenius norm. We assume we want to train a classifier $f : \mathbb{R}^d \to \{1, \dots, C\}$ to correctly assign one of C different classes to input vectors $\boldsymbol{x}$ from a d-dimensional Euclidean space. Let μ denote the probability measure on $\mathbb{R}^d$ and let $f^\star$ be an oracle that always returns the correct label for any input. The distribution μ is assumed to be of bounded support, i.e. $\mathrm{P}_{\boldsymbol{x}\sim\mu}(\boldsymbol{x} \in \mathcal{X}) = 1$ with $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \|\boldsymbol{x}\| \leq M\}$ for some $M > 0$.

Formally, adversarial perturbations are defined relative to a classifier f and an input $\boldsymbol{x}$. A perturbation $\boldsymbol{\eta}$ is called an *adversarial perturbation* of $\boldsymbol{x}$ for f if $f(\boldsymbol{x} + \boldsymbol{\eta}) \neq f(\boldsymbol{x})$ while $f^\star(\boldsymbol{x} + \boldsymbol{\eta}) = f^\star(\boldsymbol{x})$. An adversarial perturbation $\boldsymbol{\eta}$ is called *minimal* if no other adversarial perturbation $\boldsymbol{\xi}$ for $\boldsymbol{x}$ and f satisfies $\|\boldsymbol{\xi}\| < \|\boldsymbol{\eta}\|$. In this work, we will focus on minimal adversarial perturbations.

The *robustness* of a classifier f is defined as the expected norm of the smallest perturbation necessary to change the classification of an arbitrary input $\boldsymbol{x}$ sampled from μ:

$$\rho_{\mathrm{adv}}(f) = \mathrm{E}_{\boldsymbol{x}\sim\mu}[\Delta_{\mathrm{adv}}(\boldsymbol{x}; f)],$$

where

$$\Delta_{\mathrm{adv}}(\boldsymbol{x}; f) = \min_{\boldsymbol{\eta}\in\mathbb{R}^d} \{\|\boldsymbol{\eta}\| \mid f(\boldsymbol{x} + \boldsymbol{\eta}) \neq f(\boldsymbol{x})\}.$$

A *multi-index* is a tuple of non-negative integers, generally denoted by Greek letters such as α and β. For a multi-index $\alpha = (\alpha_1, \dots, \alpha_m)$ and a function f we define

$$|\alpha| = \alpha_1 + \cdots + \alpha_n, \qquad \partial^\alpha f = \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \dots \partial x_n^{\alpha_n}}.$$

The *Jacobian matrix* of a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m : \boldsymbol{x} \mapsto [f_1(\boldsymbol{x}), \dots, f_m(\boldsymbol{x})]^T$ is defined as

$$\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

3.1 Families of classifiers

A *linear classifier* is a classifier f of the form

$$f(\boldsymbol{x}) = \arg \max_{i=1,\dots,C} \boldsymbol{w}_i \cdot \boldsymbol{x} + b_i.$$

The vectors $\boldsymbol{w}_i$ are called *weights* and the scalars b_i are called *biases*.

A *Multi-Layer Perceptron* (MLP) is a classifier given by

$$\begin{aligned} f(\boldsymbol{x}) &= \arg \max_{i=1,\dots,C} \mathrm{softmax}(\boldsymbol{h}_L(\boldsymbol{x}))_i, \\ \boldsymbol{h}_L(\boldsymbol{x}) &= g_L(\boldsymbol{V}_L \boldsymbol{h}_{L-1}(\boldsymbol{x}) + \boldsymbol{b}_L), \\ &\vdots \\ \boldsymbol{h}_1(\boldsymbol{x}) &= g_1(\boldsymbol{V}_1 \boldsymbol{x} + \boldsymbol{b}_1). \end{aligned}$$

An MLP is nothing more than a series of linear transformations $\boldsymbol{V}_l \boldsymbol{h}_{l-1}(\boldsymbol{x}) + \boldsymbol{b}_l$ followed by non-linear activation functions g_l. Here, $\mathrm{softmax}$ is the softmax function:

$$\mathrm{softmax}(\boldsymbol{y})_i = \frac{\exp(\boldsymbol{w}_i \cdot \boldsymbol{y} + b_i)}{\sum_j \exp(\boldsymbol{w}_j \cdot \boldsymbol{y} + b_j)}.$$

This function is a popular choice as the final layer for an MLP used for classification, but it is by no means the only possibility. Note that having a softmax as the final layer essentially turns the network into a linear classifier of the output of its penultimate layer, $\boldsymbol{h}_L(\boldsymbol{x})$.

A *Convolutional Neural Network* (CNN) is a neural network that uses at least one convolution operation. For an input tensor $\mathbf{X} \in \mathbb{R}^{c \times d \times d}$ and a kernel tensor $\mathbf{W} \in \mathbb{R}^{k \times c \times q \times q}$, the discrete convolution of $\mathbf{X}$ and $\mathbf{W}$ is given by

$$(\mathbf{X} \star \mathbf{W})_{ijk} = \sum_{n=1}^{c} \sum_{m=1}^{q} \sum_{l=1}^{q} w_{i,n,m,l} x_{n,m+s(q-1),l+s(q-1)}.$$

Here, s is the *stride* of the convolution. The output of such a layer is a 3D tensor of size $k \times t \times t$ where $t = \frac{d-q}{s} + 1$. After the convolution operation, usually a bias $\boldsymbol{b} \in \mathbb{R}^k$ is added to each of the feature maps. The different components $(\mathbf{W} \star \mathbf{X})_i$ constitute the feature maps of this convolutional layer. In a slight abuse of notation, we will write $\mathbf{W} \star \mathbf{X} + \boldsymbol{b}$ to signify the tensor $\mathbf{W} \star \mathbf{X}$ where each of the k feature maps has its respective bias added in:

$$(\mathbf{W} \star \mathbf{X} + \boldsymbol{b})_{ijk} = (\mathbf{W} \star \mathbf{X})_{ijk} + b_i.$$

CNNs also often employ *pooling layers*, which perform a sort of dimensionality reduction. If we write the output of a pooling layer as $\mathbf{Z}(\mathbf{X})$, then we have

$$z_{ijk}(\mathbf{X}) = p(\{x_{i,n+s(j-1),m+s(k-1)} \mid 1 \leq n, m \leq q\}).$$

Here, p is the pooling operation, s is the stride and q is a parameter. The output tensor $\mathbf{Z}(\mathbf{X})$ has dimensions $c \times t \times t$. For ease of notation, we assume each pooling operation has an associated function I such that

$$z_{ijk}(\mathbf{X}) = p(\{x_{inm} \mid (n, m) \in I(j, k)\}).$$

In the literature, the set $I(j, k)$ is referred to as the *receptive field* of the pooling layer. Each receptive field corresponds to some $q \times q$ region in the input $\mathbf{X}$. Common pooling operations include taking the maximum of all inputs, averaging the inputs and taking an L_p norm of the inputs.

4 Lower bounds on classifier robustness

Comparing the architectures of several practical CNNs, it would seem the only useful approach is a "modular" one. If we succeed in lower-bounding the robustness of some layer given the robustness of the next layer, we can work our way backwards through the network, starting at the output layer and going backwards until we reach the input layer. That way, our approach can be applied to any feedforward neural network as long as the robustness bounds of the different layer types have been established. The obvious downside of this idea is that we most likely introduce cumulative approximation errors which increase as the number of layers of the network increases. In turn, however, we get a flexible and efficient framework which can handle any feedforward architecture composed of known layer types.

4.1 Softmax output layers

We now want to find the smallest perturbation $\boldsymbol{r}$ to the input $\boldsymbol{x}$ of a softmax layer such that $f(\boldsymbol{x} + \boldsymbol{r}) \neq f(\boldsymbol{x})$. It can be proven that any such perturbation satisfies

$$\|\boldsymbol{r}\| \geq \min_{c' \neq c} \frac{|(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{x} + b_{c'} - b_c|}{\|\boldsymbol{w}_{c'} - \boldsymbol{w}_c\|},$$

where $f(\boldsymbol{x}) = c$. Moreover, there exist classifiers for which this bound is tight.

4.2 Fully-connected layers

To analyze the robustness of fully-connected layers to adversarial perturbations, we assume the next layer has a robustness of κ (this will usually be the softmax output layer, however there exist CNNs which employ fully-connected layers in other locations than just at the end). We then want to find a perturbation $\boldsymbol{r}$ such that

$$\|\boldsymbol{h}_L(\boldsymbol{x} + \boldsymbol{r})\| = \|\boldsymbol{h}_L(\boldsymbol{x})\| + \kappa.$$

We find

Theorem 1. *Let $\boldsymbol{h}_L : \mathbb{R}^d \to \mathbb{R}^n$ be twice differentiable with second-order derivatives bounded by M. Then for any $\boldsymbol{x} \in \mathbb{R}^d$,*

$$\|\boldsymbol{r}\| \geq \frac{\sqrt{\|\boldsymbol{J}(\boldsymbol{x})\|^2 + 2M\sqrt{n}\kappa} - \|\boldsymbol{J}(\boldsymbol{x})\|}{M\sqrt{n}}, \qquad (1)$$

where $\boldsymbol{J}(\boldsymbol{x})$ is the Jacobian matrix of $\boldsymbol{h}_L$ at $\boldsymbol{x}$.

One can check that the assumptions on $\boldsymbol{h}_L$ are usually satisfied in practice. There also exists an efficient algorithm for approximating M, a task which otherwise might involve a prohibitively expensive optimization problem.

4.3 Convolutional layers

The next layer of the network is assumed to have a robustness bound of κ, in the sense that any adversarial perturbation $\mathbf{Q}$ to $\mathbf{X}$ must satisfy $\|\mathbf{Q}\|_F \geq \kappa$. We can now attempt to bound the norm of a perturbation $\mathbf{R}$ to $\mathbf{X}$ such that

$$\|\mathrm{ReLU}(\mathbf{W} \star (\mathbf{X} + \mathbf{R}) + \boldsymbol{b})\|_F = \|\mathrm{ReLU}(\mathbf{W} \star \mathbf{X} + \boldsymbol{b})\|_F + \kappa.$$

We find

Theorem 2. *Consider a convolutional layer with filter tensor $\mathbf{W} \in \mathbb{R}^{k \times c \times q \times q}$ and stride s whose input consists of a 3D tensor $\mathbf{X} \in \mathbb{R}^{c \times d \times d}$. Suppose the next layer has a robustness bound of κ, then any adversarial perturbation to the input of this layer must satisfy*

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{\|\mathbf{W}\|_F}. \tag{2}$$

4.4 Pooling layers

To facilitate the analysis of the pooling layers, we make the following assumption which is satisfied by the most common pooling operations:

Assumption 3. *The pooling operation satisfies*

$$z_{ijk}(\mathbf{X} + \mathbf{R}) \leq z_{ijk}(\mathbf{X}) + z_{ijk}(\mathbf{R}).$$

We have

Theorem 4. *Consider a pooling layer whose operation satisfies Assumption 3. Let the input be of size $c \times d \times d$ and the receptive field of size $q \times q$. Let the output be of size $c \times t \times t$. If the robustness bound of the next layer is κ, then the following bounds hold for any adversarial perturbation $\mathbf{R}$:*

- *MAX or average pooling:*

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{t}. \tag{3}$$

- *L_p pooling:*

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{tq^{2/p}}. \tag{4}$$

4.5 Batch Normalization layers

Batch Normalization (Ioffe et al. 2015) computes

$$\mathrm{BN}(\boldsymbol{x})_j = \gamma_j \frac{x_j - \mathrm{E}[x_j]}{\sqrt{\mathrm{Var}[x_j] + \varepsilon}} + \beta_j.$$

Here, γ_j and β_j are learnable parameters and $\varepsilon > 0$ is a parameter for numerical stability. We find

Theorem 5. *Any adversarial perturbation $\boldsymbol{r}$ to the input $\boldsymbol{x}$ of a BN layer must satisfy*

$$\|\boldsymbol{r}\| \geq \max_j \delta_j, \tag{5}$$

where

$$\begin{aligned}
\delta_j &= \min\{|\rho_{1,j}|, |\rho_{2,j}|\}, \\
\rho_{1,j} &= -\frac{\sqrt{\phi_j^2 \chi_j^2 + 4\phi_j \gamma_j \psi} + \phi_j \chi_j}{2\phi_j \gamma_j}, \\
\rho_{2,j} &= \frac{\sqrt{\phi_j^2 \chi_j^2 + 4\phi_j \gamma_j \psi} - \phi_j \chi_j}{2\phi_j \gamma_j}, \\
\phi_j &= \frac{\gamma_j}{\mathrm{Var}[x_j] + \varepsilon}, \\
\chi_j &= 2\gamma_j (x_j - \mathrm{E}[x_j]) + 2\beta_j \sqrt{\mathrm{Var}[x_j] + \varepsilon}, \\
\psi &= \frac{2\kappa \|\mathrm{BN}(\boldsymbol{x})\| + \kappa^2}{d},
\end{aligned}$$

and γ_j, β_j are the learnable parameters.

4.6 Local Response Normalization layers

Local Response Normalization (LRN), introduced by Krizhevsky et al. 2012, computes a response-normalized activity b_{ixy} given by the expression

$$b_{ixy} = a_{ixy} \left(k + \alpha \sum_{l=\max(0, i-n/2)}^{\min(N-1, i+n/2)} a_{lxy}^2 \right)^{-\beta},$$

where N is the total number of kernels in the layer and the sum runs over n "adjacent" kernel maps at the same spatial position. The constants k, n, α, β are hyperparameters of the LRN.

For an input tensor $\mathbf{X} \in \mathbb{R}^{c \times d \times d}$, the result of applying LRN to $\mathbf{X}$ is written as $\mathrm{LRN}(\mathbf{X})$. We find

Theorem 6. *Consider an LRN layer where the robustness of the next layer is given by κ. Any adversarial perturbation $\mathbf{R}$ to the input $\mathbf{X}$ of this layer must satisfy*

$$\|\mathbf{R}\|_F \geq \max_{a \in \Delta} \lambda_a, \tag{6}$$

where

$$\lambda_a = \frac{\sqrt{\|\nabla \operatorname{LRN}(\mathbf{X})_a\|^2 - 2M_a\delta_a} - \|\nabla \operatorname{LRN}(\mathbf{X})_a\|}{M_a},$$
$$\Delta = \{(i,x,y) \mid \delta_{ixy} < 0\},$$
$$\delta_{ixy} = |\operatorname{LRN}(\mathbf{X})_{ixy}| - \frac{\|\operatorname{LRN}(\mathbf{X})\|_F + \kappa}{\sqrt{d}}.$$

Again, an efficient method for computing M_a is given to avoid expensive optimization.

5 Experimental results

We tested the theoretical bounds on the MNIST and CIFAR-10 test sets using the Caffe (Jia et al. 2014) implementation of LeNet-5 (LeCun et al. 1998) as well as its built-in network for CIFAR-10, which we will call CIFAR-NIN.

Because our method only computes norms and does not provide a way to generate actual adversarial perturbations, we used the fast gradient sign method (FGS, Goodfellow et al. 2015) to adversarially perturb each sample in the test sets in order to assess the tightness of our theoretical bounds.

Table 1 shows theoretical and empirical bounds on the Frobenius norms of adversarial perturbations for LeNet-5 and CIFAR-NIN. These bounds were computed by averaging over all samples in their respective test sets (MNIST for LeNet-5 and CIFAR-10 for CIFAR-NIN) and then normalized by dividing by the number of pixels in the input (i.e. 28×28 for LeNet and $3 \times 32 \times 32$ for CIFAR-NIN). The table also shows the percentage of samples we were able to adversarially perturb using FGS.

6 Improving the robustness

Consider a network consisting solely of convolutional and pooling layers. Suppose each layer has an original robustness of κ, but we succeed in increasing the robustness of layer i by a quantity δ. For simplicity, we assume this increase is the same in each layer. Restricting our attention to the convolutional and pooling layers, we find

$$\frac{\kappa_1'}{\kappa_1} = 1 + \delta \sum_{l=1}^{L} \frac{1}{\kappa_l}. \tag{7}$$

Here, κ_i is the robustness of layer i of the original network and κ_1' is the robustness of the modified network. If we assume $\kappa_i = \alpha\kappa_{i+1}$ for all $i < L$ and $\alpha \neq 1$, then $\kappa_i = \alpha^{L-i}\kappa_L$ and

$$\frac{\kappa_1'}{\kappa_1} = 1 + \frac{\delta(1-\alpha^{-L})}{\kappa_L(1-\alpha^{-1})}.$$

This relationship is linear in δ but exponential in L as long as $0 < \alpha < 1$. Figure 1 shows a plot of the relative increase in theoretical robustness as a function of the number of layers for $\alpha = 0.01$, $\delta = 0.1$ and $\kappa_L = 2$. For $\alpha > 1$, the robustness can still be increased, but the effect will saturate at a certain number of layers. The relative increase at this saturation point is given by

$$\lim_{L\to\infty} 1 + \frac{\delta(1-\alpha^{-L})}{\kappa_L(1-\alpha^{-1})} = 1 + \frac{\delta}{\kappa_L(1-\alpha^{-1})}.$$

We conclude that it is possible to significantly increase the robustness of a neural network by only slightly increasing the robustness of its individual layers. Moreover, this effect grows exponentially in the number of layers. Thus, although the linearity hypothesis indicates that deeper networks are more vulnerable to adversarial perturbations, our analysis suggests that it is actually easier to increase the robustness of a deep network than the robustness of a shallow one.

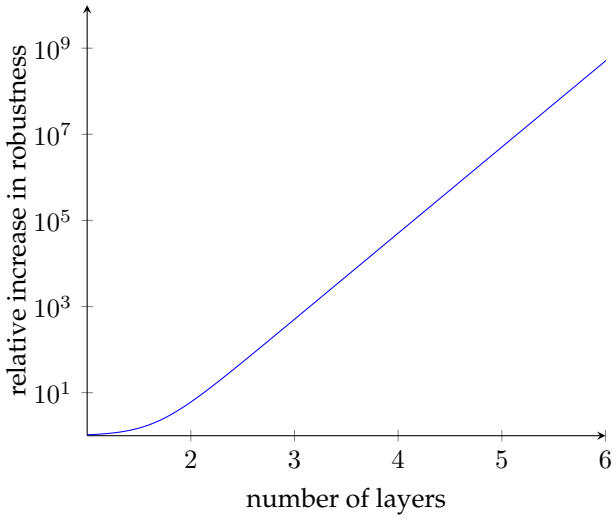


Figure 1: The relative increase in theoretical robustness of a CNN as a function of the number of layers, where one increases the robustness of each individual layer by $\delta = 0.1$ and the output layer has a robustness of 2. It is assumed that $\kappa_l = 0.01\kappa_{l+1}$ for all $l < L$.

Network	Normalized theoretical bound	Normalized empirical bound	Rate
LeNet-5	$7.274294\text{e}{-8}$	0.9334476	99.94%
CIFAR-NIN	$1.579417\text{e}{-17}$	0.3129927	93.04%

Table 1: Theoretical and empirical robustness bounds on different convolutional networks

6.1 Experimental results

In this section, we explore the effect of adding Local Response Normalization layers on the robustness of LeNet-5. Specifically, we create three variants of the original LeNet-5:

- LeNet-LRN1. This is LeNet-5 with a single LRN layer after the first pooling operation.
- LeNet-LRN2. This is LeNet-5 with a single LRN layer after the second pooling operation.
- LeNet-LRN3. This is LeNet-5 with two LRN layers, one after each pooling operation.

The results are summarized in Table 2. From these results, we can conclude that the introduction of LRN layers into the network can indeed lower the adversarial error rate as well as increase the robustness of the network. However, the precise location of the LRN layers plays an important role. Specifically, the results seem to indicate that LRN layers placed farther towards the end of the network have a diminished effect on robustness and may even cause the robustness to decrease. LeNet-LRN1 and LeNet-LRN3 have a significantly larger minimal perturbation, though, and all networks have a lower adversarial error rate than the original LeNet-5. The accuracy also does not seem to have suffered much from these modifications, suggesting that normalization layers (LRN, BN or otherwise) are a viable technique for increasing robustness without sacrificing accuracy. On the other hand, the diminishing returns suggested by the theoretical analysis appear to hold in this case: the first LRN layer in LeNet-LRN1 increases the minimal ε from 0.00065 to 0.00756. The addition of the second LRN layer in LeNet-LRN3 yields a minimal ε of 0.00869, which is a much smaller increase than we obtained using the first layer.

7 Conclusion and future work

In this work, we have attempted to characterize the precise nature and cause of adversarial perturbations. This was done by deriving lower bounds on the magnitude of the minimal perturbations necessary to change the classification of a given classifier. Whereas other related works such as Fawzi et al. 2016 do this by considering general classifiers $f : \mathbb{R}^d \to \{1, \ldots, C\}$, here we have focused our attention on specific families of classifiers and attempted to relate their robustness directly to their model parameters and hyperparameters. We studied linear classifiers, Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). Due to the broad range of architectures used in modern CNNs, the analysis for these networks was performed in a modular manner: we considered each layer in isolation and derived lower bounds on the robustness of a layer given only the robustness of the next layer. We empirically verified the theoretical bounds on every family of classifiers and found no violations. The results of our analysis seem to suggest that it is possible to significantly increase the robustness of a neural network by slight increments in the robustness of its individual layers, and this effect grows exponentially with the number of layers. Furthermore, we find that the introduction of normalization layers has a positive effect on robustness without sacrificing accuracy.

The proposed bounds can be computed efficiently, requiring an amount of operations that grows linearly with the number of model parameters and hyperparameters. This makes them useful for several applications, such as model selection and constructing techniques for improving robustness.

The theoretical analysis in this work was limited to linear classifiers, Multi-Layer Perceptrons with softmax output layers and Convolutional Neural Networks consisting of convolutional layers, pooling layers, Batch Normalization layers and Local Response Normalization layers. The most obvious avenue for future work is to extend this analysis to other types of layers as well as other architectures, such as recurrent networks.

Another avenue is to fine-tune the bounds so that they deviate less from the actual perturbations found by methods such as FGS, since the bounds

Network	Accuracy	Min	Max	Mean	Median	Std	Rate
LeNet-LRN1	98.76%	0.00756	99.71930	25.31045	21.88714	17.30748	98.36%
LeNet-LRN2	98.48%	0.00011	99.87615	24.92861	19.62794	20.60854	95.52%
LeNet-LRN3	98.94%	0.00869	99.95050	25.76302	20.96658	18.83517	96.94%

Table 2: Summary of ε values for the LeNet-LRN networks

proposed here do not appear to be very tight. There are also no error estimates given for the proposed theoretical bounds. Quantifying how much the bounds can deviate from the actual perturbations could improve the estimates and make the bounds tighter.

Formally characterizing the relationship between accuracy and robustness is a question we definitely also want to explore further. We expect that the robustness of a classifier can be increased without loss of accuracy (or even with an increase in accuracy) up to a point where the robustness equals that of the oracle $f^\star$; increasing the robustness beyond this point should cause the accuracy to decrease as the classifier becomes too insensitive to changes.

Finally, we think it would also be worthwhile to study the trade-off between neural network depth and breadth. It is strongly suspected that deeper networks can be exponentially more efficient than shallow ones, but the precise effect of increasing depth vs. increasing breadth on the robustness has not been studied yet to our knowledge.

References

Fawzi, Alhussein, Seyed-Mohsen Moosavi-Dezfooli and Pascal Frossard (2016). 'Robustness of classifiers: from adversarial to random noise'. In: *Advances in Neural Information Processing Systems*, pp. 1624–1632.

Goodfellow, Ian J, Jonathon Shlens and Christian Szegedy (2015). 'Explaining and Harnessing Adversarial Examples'. In: *Iclr 2015*, pp. 1–11.

Ioffe, Sergey and Christian Szegedy (2015). 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *Journal of Machine Learning Research*. DOI: 10.1007/s13398-014-0173-7.2.

Jia, Yangqing et al. (2014). 'Caffe: Convolutional architecture for fast feature embedding'. In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, pp. 675–678.

Krizhevsky, Alex, IIya Sulskever and Geoffret E Hinton (2012). 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information and Processing Systems (NIPS)*, pp. 1–9.

LeCun, Yann et al. (1998). 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Szegedy, Christian, W Zaremba and I Sutskever (2013). 'Intriguing properties of neural networks'. In: *arXiv preprint*, pp. 1–10. ISSN: 15499618. DOI: 10.1021/ct2009208. arXiv: arXiv:1312.6199v4. URL: http://arxiv.org/abs/1312.6199.

Contents

Summary vii

1 Introduction 1
- 1.1 Adversarial perturbations . . . 2
- 1.2 Goal of this thesis . . . 3
- 1.3 Outline . . . 4

2 Preliminaries 5
- 2.1 Notation . . . 5
- 2.2 Definitions . . . 5
- 2.3 Theoretical framework . . . 6
  - 2.3.1 Adversarial perturbations . . . 6
- 2.4 Families of classifiers . . . 7
  - 2.4.1 Linear classifier . . . 8
  - 2.4.2 Multi-Layer Perceptron . . . 9
  - 2.4.3 Convolutional Neural Networks . . . 11
- 2.5 Generating adversarial perturbations . . . 13
  - 2.5.1 L-BFGS . . . 13
  - 2.5.2 Fast gradient sign . . . 14
  - 2.5.3 Hot/cold . . . 14
  - 2.5.4 Fast gradient value . . . 15
  - 2.5.5 Evaluation . . . 15
- 2.6 Protections against adversarial perturbations . . . 16
  - 2.6.1 Deep Contractive Network . . . 17
  - 2.6.2 Generative Adversarial Networks . . . 17
  - 2.6.3 Fast Gradient Sign . . . 17
  - 2.6.4 Foveations . . . 17
  - 2.6.5 Dimensionality reduction . . . 18
  - 2.6.6 Stability training . . . 18
  - 2.6.7 Batch-Adjusted Network Gradients . . . 18
  - 2.6.8 Hot/cold . . . 18
  - 2.6.9 Defensive distillation . . . 18
- 2.7 Iterative perturbation . . . 19

3 Linear classifiers 21
- 3.1 Binary classification . . . 21
- 3.2 Multiclass classification . . . 23
  - 3.2.1 Computing the bound . . . 25
  - 3.2.2 Tightness of the bound . . . 25
- 3.3 Experiments . . . 25

3.3.1 Discussion . . . 26

4 Multi-Layer Perceptrons **29**
4.1 Robustness . . . 29
4.1.1 One-layer MLP . . . 29
4.1.2 L-layer MLP . . . 30
4.2 Realism of the assumptions . . . 31
4.3 Experiments . . . 34
4.3.1 M parameter . . . 34
4.3.2 Results . . . 35
4.3.3 Discussion . . . 38
4.4 Improving the robustness of MLPs . . . 39

5 Convolutional Neural Networks **41**
5.1 Practical examples . . . 41
5.1.1 LeNet-5 . . . 41
5.1.2 CIFAR-NIN . . . 41
5.1.3 AlexNet . . . 43
5.1.4 VGGNet . . . 43
5.1.5 GoogLeNet . . . 44
5.1.6 ResNet . . . 44
5.2 Robustness of CNNs to adversarial perturbations . . . 44
5.2.1 Convolutional layers . . . 47
5.2.2 Pooling layers . . . 48
5.2.3 Batch Normalization layers . . . 50
5.2.4 Local Response Normalization layers . . . 53
5.2.5 Flattening . . . 58
5.2.6 Computing the robustness bounds of a CNN . . . 58
5.3 Experimental results . . . 58
5.3.1 Discussion . . . 59
5.4 Improving the robustness of CNNs . . . 63
5.4.1 Experimental results . . . 66

6 Conclusion **69**
6.1 Future work . . . 69

A Auxiliary results **71**
A.1 Multi-Layer Perceptrons . . . 71
A.2 Convolutional Neural Networks . . . 71

Bibliography **75**

List of Figures

1.1 An example of an imperceptible perturbation causing a misclassification on GoogLeNet. . . . 3

2.1 Some examples of commonly used activation functions. . . . 10
2.2 Graphical depiction of an MLP . . . 11
2.3 A typical CNN architecture. . . . 12
2.4 Example of adversarial perturbations from Szegedy et al. 2013 . . . 16
2.5 Iterative perturbation . . . 19

3.1 Minimal adversarial perturbation for a binary classifier . . . 23
3.2 Boxplots of bound deviations for linear classifiers . . . 27

4.1 Boxplots of average bound deviations for MLP tests . . . 37
4.2 Boxplots of deviations of adversarial perturbations from the theoretical bound in trained MLPs 38
4.3 Accuracy, error rate and minimal adversarial perturbations when regularizing only the output layer. . . . 40

5.1 Some MNIST samples . . . 42
5.2 Schematic diagram of LeNet-5 architecture. . . . 42
5.3 Some CIFAR-10 samples . . . 43
5.4 AlexNet architecture. . . . 44
5.5 GoogLeNet architecture. . . . 45
5.6 Inception module with dimensionality reductions . . . 46
5.7 Shortcut connections in ResNet. . . . 46
5.8 Selection of adversarial examples for LeNet-5 . . . 60
5.9 Selection of adversarial examples for CIFAR-NIN . . . 61
5.10 Heatmap of adversarial perturbations on the different CNNs . . . 62
5.11 Boxplots of ε values for the different CNNs . . . 62
5.12 Relative increase in theoretical robustness obtained by scaling kernel tensors . . . 65
5.13 Relative increase in theoretical robustness obtained by increasing per-layer robustness . . . 66

List of Tables

3.1 Test results on linear classifiers 26

4.1 Overview of bounds for first and second derivatives of some activation functions 35
4.2 Test results on random MLPs 36
4.3 Trained MLP statistics of bound deviations 38

5.1 Theoretical and empirical robustness bounds on different convolutional networks 59
5.2 Statistics of ε values for the different CNNs 62
5.3 Summary of ε values for the LeNet-LRN networks 67

Chapter 1

Introduction

Historically, computers have had the reputation of being able to solve problems that can be described by formal, mathematical rules, but being unable to (or at least very ineffective at) solving tasks which do not admit a concise formal description. Computers can, for example, flawlessly perform arithmetical operations on large numbers in a matter of milliseconds, whereas those same operations would take a human being several hours or even days (if completed at all). On the other hand, tasks which humans perform nearly instantaneously can prove very difficult for machines to carry out, such as recognizing objects in pictures or translating one language into another. The crucial distinction between these two categories of tasks appears to be *context*. One does not need much knowledge about the outside world in order to play a good game of chess, but in order to translate a sentence from Russian to English, for instance, a great deal of contextual knowledge may be needed. The field of Artificial Intelligence (AI) was born out of the desire to make machines be as proficient at such "fuzzy", informal tasks as they are at formal, mathematically precise tasks. Early results in the 1950s and 1960s were achieved mainly on contrived, toy problems which often required the manual specification of the complete set of background information necessary for finding the solution. Examples of such results include SHRDLU (Winograd 1972) and the General Problem Solver (GPS, Newell et al. 1959). Though initially viewed as promising, these approaches quickly break down when the amount of necessary contextual information increases to realistic proportions, usually because it becomes infeasible to encode all the information in the formal language understood by the system or because the problem simply becomes too large for any conceivable machine to handle efficiently.

Since the late 1990s and early 2000s, a sort of paradigm shift has taken place in AI which paved the way for the considerable successes the field has enjoyed in recent years. Whereas the emphasis used to lie on what *algorithm* to apply to solve a given AI problem, nowadays researchers are more concerned with what *data* to use. The algorithmic considerations, though still important, have taken a back seat thanks to the availability of very large data sets like ImageNet (Deng et al. 2009), which is a hierarchical database containing over ten million annotated color images; or the Human Genome Project which maintains a data set of billions of base pairs of genomic sequences (Collins et al. 2003). These large data sets fueled the development of a subfield of AI called Machine Learning (ML). The core idea underpinning the field of ML is to design statistical models which can be "trained" to automatically extract useful features from input data, and to use those features to solve AI problems such as object recognition or machine translation. It is now no longer strictly required to manually encode the relevant contextual information into the AI program before it can begin to solve the problem at hand. Instead, ML models "learn" by themselves what information to extract from their data sets in order to best solve the given problem. The success of this approach is heavily predicated on the availability of a sufficient amount of data (usually far in excess of what an average human being would need for the same task), as well as the ability to actually process it in a reasonably efficient manner. The fulfillment of these two conditions is the reason ML ever became as popular as it is now.

It probably comes as no big surprise that most of the models used in ML draw inspiration from the mammalian brain. The earliest example of this is the Perceptron (Rosenblatt 1958), which aimed to model how

information is stored and processed within the human brain. By linking many Perceptrons together and creating a so-called Multi-Layer Perceptron (MLP), one can in principle approximate any function to any desired degree of accuracy (Hornik 1991). These findings resulted in the birth of a class of models known as Artificial Neural Networks (ANNs), which have become very popular models in ML (though they are by no means the only models in use). Later works such as the Cognitron (Fukushima 1975) and especially the Neocognitron (Fukushima 1988) played a major role in the development of so-called Convolutional Neural Networks (CNNs). They have been particularly effective at object recognition tasks, as first demonstrated by LeCun et al. 1998a who created LeNet-5, a CNN which outperformed all its competitors at the time in the task of recognizing handwritten digits. CNNs again showed great promise compared to other existing techniques when Krizhevsky et al. 2012a won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012 by a significant margin with their model, AlexNet. In this challenge, researchers from all over the world competed to create the best algorithm for object recognition and image classification on the ImageNet data set. Since then, there has been a surge of interest in ML, both academically and commercially. The models proposed in the literature have steadily increased in complexity, leading to the coinage of the term *deep learning* to describe these approaches to ML. Curiously, the size of ML models appears to double approximately every two years, just like Moore's Law predicts is the case for the number of components on integrated circuits (Moore 1998). With their increased size of course also comes increased capacity to solve more complex tasks. The Perceptron could only be used to distinguish two different categories of objects; LeNet-5 can recognize ten different handwritten digits; AlexNet can recognize 1,000 different classes of objects. Aside from classifying objects into categories, neural networks are now also being trained to play video games (Mnih et al. 2015) or even to produce entire natural-language descriptions of images (You et al. 2016).

1.1 Adversarial perturbations

Despite their big successes in various AI tasks, there of course remain plenty of problems to be solved. Perhaps the biggest problem facing ML research is the fact that many of the proposed models are basically black boxes: no one can explain exactly why they work or even tell exactly what they're doing to their input data most of the time. This has led to the identification of several unexpected and counter-intuitive properties of neural networks. In particular, Szegedy et al. 2013 discovered that the input-output mappings learned by state-of-the-art neural networks are significantly discontinuous. It is possible to cause a neural network used for image recognition to misclassify its input by applying a very specific, hardly perceptible perturbation to the input. Goodfellow et al. 2015 illustrate this phenomenon with Figure 1.1. They start with a picture of a panda from ImageNet, which is correctly classified by GoogLeNet (Szegedy et al. 2015). Then, they apply a perturbation so small it is imperceptible to the human eye, yet the neural network changes its classification to "gibbon" with over 99% confidence. Szegedy et al. 2013 call these perturbed inputs "adversarial examples".

Since the puzzling discovery of adversarial perturbations, several hypotheses have been proposed to explain why they exist (Goodfellow et al. 2015; Lou et al. 2016; Rozsa et al. 2016b), but none of these have actually been proven so far. A number of methods have also been proposed to make models more robust to these perturbations (Goodfellow et al. 2015; Goodfellow et al. 2014; Gu et al. 2014; Lou et al. 2016; Maharaj 2016; Papernot et al. 2016; Rozsa et al. 2016a,b; Zheng et al. 2016), but none have been able to solve the problem entirely. One may wonder whether all this attention to adversarial examples is warranted. After all, one might object that these samples are rare, contrived and would never occur in practice. However, there are several reasons adversarial examples are important. Goodfellow et al. 2015 discovered that these samples generalize across different models. A specific adversarial example that was designed to fool a certain model A will often also fool a different model B. This happens with high probability even if both models have different architectures or are trained on different data sets. Moosavi-Dezfooli et al. 2016 take this one step further and demonstrate that the adversarial *perturbations* themselves, not the adversarial examples resulting from adding such perturbations to existing images, can generalize across different models and data sets with high probability. Furthermore, Kurakin et al. 2016 find that

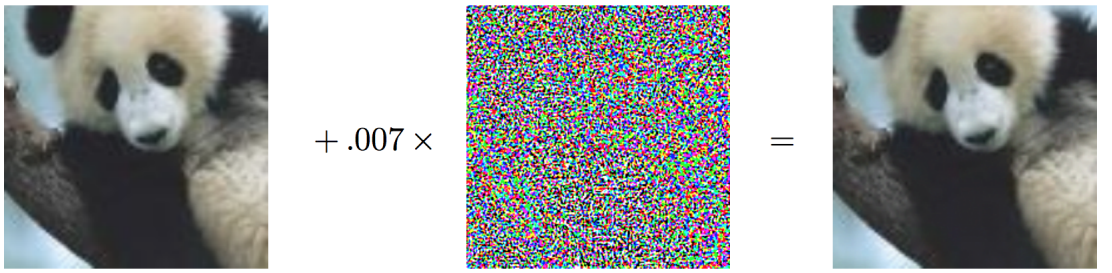


Figure 1.1: An example of an imperceptible perturbation causing a misclassification on GoogLeNet.

adversarial examples can occur in the physical world and thus are not as contrived as may seem at first glance.

These findings lead one to question the generalization capabilities of modern neural networks. If neural network classifiers really fully "understood" the tasks that are asked of them, then obviously they should be impervious to adversarial perturbations. The fact that they are not is an indication that something may be fundamentally wrong. This is illustrated in Ribeiro et al. 2016, where the authors propose a technique called Local Interpretable Model-agnostic Explanations (LIME) that can be used to explain the predictions of classifiers. LIME does this by highlighting the parts of an input the model considers most important in making its prediction. In their work, the authors provide an example of two classifiers, A1 and A2, that are tasked with determining whether or not a person is religious based on documents they have written. Using LIME, the authors demonstrate that the classifier with the highest accuracy on the test set, A2, actually performs worse than A1, because A2 bases its predictions on words that have nothing to do with religion. A1, on the other hand, bases its predictions on words that are relevant to religion. Thus, the good performance of A2 on the test set is due merely to spurious correlations in the data that do not generalize. Other results such as Dauphin et al. 2013 and Zhang et al. 2016 also suggest that large neural networks have trouble effectively utilizing their capacity and generalizing well.

The existence of adversarial perturbations also calls into question the security of classifiers. Given the fact that these perturbations apparently generalize to other models and other data sets besides the ones they were specifically designed for, a simple attack can be devised. Consider, for example, a mature content filter meant to detect pornographic content uploaded to an image hosting service. This filter is basically a binary classifier which takes an image as input and outputs 1 or 0 depending on whether it detected pornographic content or not. If we wish to fool this classifier to circumvent the filter, we could train our own classifier on a data set we curated ourselves and generate adversarial examples for that classifier. With high probability, the adversarial examples that fooled our classifier will also fool whatever classifier is running on the production servers we are attacking, even though their architectures and data sets may be totally different. Since adversarial examples generally do not look any different from legitimate inputs, these attempts at fooling the classifier could go completely unnoticed by human moderators.

1.2 Goal of this thesis

It should be clear from the above discussion that adversarial examples pose a real threat. Understanding why they exist and how to lessen their impact on ML models is therefore a worthwhile undertaking. Unfortunately, very little research has been done so far regarding the nature of adversarial perturbations and the precise reason why classifiers are so sensitive to them. Several hypotheses have been proposed, but none

have been proven; several techniques for improving the robustness to adversarial perturbations have been described, but they either show limited success or suffer from large computational overhead. Moreover, these techniques are usually based on unproven hypotheses about the nature of adversarial perturbations. This is unfortunate, as a rigorous theoretical analysis of adversarial perturbations may not only provide insight into why they exist, but also guide the design of a proven method to remedy them.

The goal of this thesis, then, is to formally characterize adversarial perturbations. We study several different families of classifiers and prove lower bounds on the size of the perturbations necessary to make them change their classification. Based on this analysis, we make some recommendations which may improve classifier robustness. It is hoped that analyses such as this will provide the insight necessary to improve the robustness of classifiers to adversarial perturbations without sacrificing their effectiveness.

1.3 Outline

Chapter 2 describes the theoretical framework used throughout this thesis as well as other prerequisites the reader may be unfamiliar with. Chapters 3 to 5 each treat a specific family of classifiers:

- Chapter 3 analyzes linear classifiers;
- Chapter 4 analyzes Multi-Layer Perceptrons;
- Chapter 5 analyzes Convolutional Neural Networks.

These chapters all have the same basic layout. First, we attempt to formally characterize the robustness of these classifiers to adversarial perturbations by deriving lower bounds on the size of perturbations necessary to change the classification of an input. Each chapter ends with empirical experiments validating the bounds. Finally, Chapter 6 concludes the work and provides some ideas for future work.

Chapter 2

Preliminaries

2.1 Notation

Throughout this thesis, scalars and elements of discrete sets will be written in cursive: $x \in \mathbb{R}$ and $y \in \{1, \ldots, C\}$. Vectors will be written in cursive boldface: $\boldsymbol{x} \in \mathbb{R}^d$. Matrices are written just like vectors, but always in uppercase: $\boldsymbol{M} \in \mathbb{R}^{m \times n}$. Tensors are written in sans-serif boldface: $\mathsf{T} \in \mathbb{R}^{l \times m \times n}$. The notation $\boldsymbol{w} \cdot \boldsymbol{x}$ will mean the inner product of $\boldsymbol{w}$ and $\boldsymbol{x}$. The multiplication of a matrix $\boldsymbol{A}$ with a vector $\boldsymbol{x}$ is written $\boldsymbol{A}\boldsymbol{x}$. The transpose of a matrix $\boldsymbol{A}$ is $\boldsymbol{A}^T$. For a vector $\boldsymbol{x}$, we denote by $\|\boldsymbol{x}\|_p$ the L_p norm of $\boldsymbol{x}$:

$$\|\boldsymbol{x}\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}.$$

If p is omitted, we mean the Euclidean norm $\|\cdot\|_2$.

The probability of an event A occuring is written as $\mathrm{P}(A)$. In case A contains a variable x which is sampled according to some distribution μ, this will be made clear by writing $\mathrm{P}_{x \sim \mu}(A)$.

2.2 Definitions

The *Frobenius norm* of a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is defined as

$$\|\boldsymbol{X}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |x_{ij}|^2}.$$

The Frobenius norm can be extended to tensors $\mathsf{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ as well in the obvious way:

$$\|\mathsf{X}\|_F = \sqrt{\sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} |x_{i_1 \ldots i_d}|^2}.$$

For any matrix $\boldsymbol{A}$, the induced Euclidean matrix norm is defined by

$$\|\boldsymbol{A}\| = \sup_{\boldsymbol{x} \neq 0} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}. \tag{2.1}$$

A *multi-index* is a tuple of non-negative integers, generally denoted by Greek letters such as α and β. For a multi-index $\alpha = (\alpha_1, \ldots, \alpha_m)$ and a function f we define

$$|\alpha| = \alpha_1 + \cdots + \alpha_n, \qquad \partial^\alpha f = \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \ldots \partial x_n^{\alpha_n}}.$$

The *Jacobian matrix* of a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m : \boldsymbol{x} \mapsto [f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})]^T$ is defined as

$$\boldsymbol{J_f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

For functions $f : \mathbb{R}^n \to \mathbb{R}$ which output scalars, the Jacobian matrix is just a vector, which we call the *gradient* and write as $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ or just $\nabla f(\boldsymbol{x})$ when no confusion can arise.

2.3 Theoretical framework

The theoretical framework used throughout this thesis is that of Fawzi et al. 2016 and Papernot et al. 2016. We assume we want to train a classifier $f : \mathbb{R}^d \to \{1, \ldots, C\}$ to correctly assign one of C different classes to input vectors $\boldsymbol{x}$ from a d-dimensional Euclidean space. Let μ denote the probability measure on $\mathbb{R}^d$ and let $f^\star$ be an oracle that always returns the correct label for any input. The distribution μ is assumed to be of bounded support, i.e. $\mathrm{P}_{\boldsymbol{x}\sim\mu}(\boldsymbol{x} \in \mathcal{X}) = 1$ with $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \|\boldsymbol{x}\| \leq M\}$ for some $M > 0$.

The robustness of classifiers to adversarial perturbations is quantified as follows. Let $\Delta_{\mathrm{adv}}(\boldsymbol{x}; f)$ denote the norm of the smallest perturbation that changes the classification of $\boldsymbol{x}$ by f:

$$\Delta_{\mathrm{adv}}(\boldsymbol{x}; f) = \min_{\boldsymbol{r}\in\mathbb{R}^d} \{\|\boldsymbol{r}\| \mid f(\boldsymbol{x}) \neq f(\boldsymbol{x} + \boldsymbol{r})\}. \tag{2.2}$$

The robustness of f to adversarial perturbations is defined as the expected value of $\Delta_{\mathrm{adv}}(\boldsymbol{x}; f)$ over all choices of $\boldsymbol{x}$ according to μ:

$$\rho_{\mathrm{adv}}(f) = \mathrm{E}_{\boldsymbol{x}\sim\mu}[\Delta_{\mathrm{adv}}(\boldsymbol{x}; f)]. \tag{2.3}$$

The accuracy of f is defined as the probability that f classifies a sample $\boldsymbol{x}$ from μ correctly:

$$\alpha(f) = \mathrm{P}_{\boldsymbol{x}\sim\mu}(f(\boldsymbol{x}) = f^\star(\boldsymbol{x})).$$

2.3.1 Adversarial perturbations

An explicit formal definition of adversarial perturbations is certainly in order, since they constitute the main object of study in this thesis. We have already discussed what we mean informally with this term. Formally, adversarial perturbations are defined relative to a classifier f and an input $\boldsymbol{x}$. A perturbation $\boldsymbol{\eta}$ is called an *adversarial perturbation* of $\boldsymbol{x}$ for f if the following conditions hold:

1. $\boldsymbol{\eta}$ changes the classification of f: $f(\boldsymbol{x} + \boldsymbol{\eta}) \neq f(\boldsymbol{x})$;

2. the correct classification of $\boldsymbol{x} + \boldsymbol{\eta}$ should be the same as that of $\boldsymbol{x}$: $f^\star(\boldsymbol{x} + \boldsymbol{\eta}) = f^\star(\boldsymbol{x})$.

An adversarial perturbation $\boldsymbol{\eta}$ is called *minimal* if it is also the case that no other adversarial perturbation $\boldsymbol{\xi}$ exists for $\boldsymbol{x}$ and f such that $\|\boldsymbol{\xi}\| < \|\boldsymbol{\eta}\|$. In this work, we will mostly focus on minimal adversarial perturbations unless otherwise mentioned.

Perturbations satisfying the definition used above would be called *virtual adversarial perturbations* by some authors such as Miyato et al. 2015. This is to emphasize the fact that the classification of $\boldsymbol{x}$ or $\boldsymbol{x} + \boldsymbol{\eta}$ by f need not be right or wrong; $\boldsymbol{\eta}$ simply has to change the classification even though both samples should belong to the same class. A more strict definition of adversarial perturbations would then also require $f(\boldsymbol{x}) = f^\star(\boldsymbol{x})$, but we will not use that definition here. Since our aim is to focus more on the robustness itself, the accuracy of the classifier is not a primary concern.

2.4 Families of classifiers

Each classifier we consider here will belong to a certain family of classifiers and thus it will have a concrete form parameterized by a set of parameters θ. The goal of machine learning is to find settings for each parameter in θ such that the accuracy of the classifier is maximized. This is usually done by taking a set of N i.i.d. samples $\mathcal{D} = \{(\boldsymbol{x}_i, c_i) \mid i = 1, \ldots, N\}$ from μ and defining a continuous loss function $\mathcal{L}$ which depends on θ and $\mathcal{D}$. The loss function is constructed so that when θ is optimal, $\mathcal{L}$ reaches its global minimum on $\mathcal{D}$. Training the classifier f then amounts to minimizing $\mathcal{L}$ via any appropriate optimization method, usually some form of gradient descent (Goodfellow et al. 2016).

A commonly used method for constructing loss functions is the *negative log-likelihood* (NLL) of the model. This is based on the idea that our choice of model and parameter set constitutes a hypothesis explaining how the samples from the different classes can be distinguished from each other. According to Bayes' theorem, the probability that our hypothesis θ is correct based on the data $\mathcal{D}$ that we have seen can be written as

$$\mathrm{P}(\theta \mid \mathcal{D}) = \frac{\mathrm{P}(\theta)\mathrm{P}(\mathcal{D} \mid \theta)}{\mathrm{P}(\mathcal{D})}.$$

The distributions $\mathrm{P}(\theta \mid \mathcal{D})$, $\mathrm{P}(\theta)$ and $\mathrm{P}(\mathcal{D} \mid \theta)$ are known respectively as the posterior, the prior and the likelihood (Murphy 2012). We wish to find the settings for θ such that $\mathrm{P}(\theta \mid \mathcal{D})$ is maximized:

$$\theta^{\mathrm{MAP}} = \arg\max_{\theta} \mathrm{P}(\theta \mid \mathcal{D}) = \arg\max_{\theta} \mathrm{P}(\theta)\mathrm{P}(\mathcal{D} \mid \theta).$$

The last equality follows from the fact that $\mathrm{P}(\mathcal{D})$ is independent of θ. Also, since $\mathcal{D}$ is assumed to consist of i.i.d. samples from μ, we can factorize $\mathrm{P}(\mathcal{D} \mid \theta)$:

$$\mathrm{P}(\mathcal{D} \mid \theta) = \prod_{i=1}^{N} \mathrm{P}(c_i \mid \boldsymbol{x}_i, \theta).$$

We then have

$$\theta^{\mathrm{MAP}} = \arg\max_{\theta} \mathrm{P}(\theta) \prod_{i=1}^{N} \mathrm{P}(c_i \mid \boldsymbol{x}_i, \theta).$$

The result of this optimization problem is called the *maximum a posteriori estimate* (MAP estimate) of θ, because it maximizes the posterior probability. To aid numerical stability when actually computing this estimate, we always take logarithms:

$$\theta^{\mathrm{MAP}} = \arg\max_{\theta} \log \mathrm{P}(\theta) + \sum_{i=1}^{N} \log \mathrm{P}(c_i \mid \boldsymbol{x}_i, \theta).$$

This transformation does not change the position of the optimum mathematically, but numerically it can avoid pathological rounding errors when multiplying many probabilities together. Especially nowadays, data sets $\mathcal{D}$ tend to be very large, and directly taking the product of the likelihoods of all the items would almost always produce zero in standard IEEE 754 floating point.

The MAP estimate has the curious property that it depends on a prior, $\mathrm{P}(\theta)$. This distribution expresses prior beliefs about what forms θ can take. Historically, prior distributions have been controversial in Bayesian statistics (Efron 1986; Gelman 2008) because in many cases, we cannot objectively defend any prior beliefs about θ at all, and a wrong choice of prior might cause a completely wrong model to be fitted. A different method called *maximum likelihood estimation* (MLE) avoids the use of a prior by maximizing the likelihood of θ instead of its posterior (again, using logarithms for numerical stability):

$$\theta^{\mathrm{MLE}} = \arg\max_{\theta} \sum_{i=1}^{N} \log \mathrm{P}(c_i \mid \boldsymbol{x}_i, \theta).$$

Though controversial in more "practical" settings such as experimental physics, priors are routinely used in machine learning for regularization. The idea is that MLE is prone to overfitting, where the model will fail to generalize to unseen samples from μ because it pays too much attention to spurious correlations or other random artifacts within the data set. ML researchers frequently use specific priors to combat these problems. For example, we may assume the parameters of the model (whatever they are) are normally distributed with zero mean and unit variance:

$$\mathrm{P}(\theta) \sim \mathcal{N}(0,1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right).$$

Taking logs and leaving out terms which are irrelevant to θ (as we would in MAP estimation), we obtain

$$\log \mathrm{P}(\theta) \propto -\sum_{i=1}^{k} \theta_i^2,$$

where we have assumed $\theta = \{\theta_1, \ldots, \theta_k\}$. In the MAP estimation, this corresponds to maximizing the likelihood $\mathrm{P}(\mathcal{D} \mid \theta)$ while at the same time minimizing the sum of squares of the parameters. This is a popular form of regularization called L_2 regularization or weight decay (Connect et al. 1992; Murphy 2012).

When using NLL, the loss function of the model is simply the negative of the log of the MLE:

$$\mathcal{L}(\mathcal{D}, \theta) = -\sum_{i=1}^{N} \log \mathrm{P}(c_i \mid \boldsymbol{x}_i, \theta).$$

Minimizing this function clearly yields the MLE for θ. When regularizing a model with some term $\Omega(\theta)$ such as weight decay, this is subtracted from the loss function, basically yielding a MAP estimate whose prior is governed by $\Omega(\theta)$:

$$\mathcal{L}(\mathcal{D}, \theta) = -\Omega(\theta) - \sum_{i=1}^{N} \log \mathrm{P}(c_i \mid \boldsymbol{x}_i, \theta).$$

2.4.1 Linear classifier

The first familiy of classifiers we will study is the family of linear classifiers. A linear classifier bases its decision on a linear combination of input elements. In general, such a classifier is parameterized by a *weight matrix* $\boldsymbol{W} \in \mathbb{R}^{C \times d}$ as well as a *bias vector* $\boldsymbol{b} \in \mathbb{R}^C$. To classify an input sample $\boldsymbol{x}$, a linear classifier computes

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}.$$

The vector $\boldsymbol{y}$ has C components, each corresponding to a certain class. The component of $\boldsymbol{y}$ with the highest value is taken to be the class of $\boldsymbol{x}$:

$$f(\boldsymbol{x}) = \arg\max_i y_i(\boldsymbol{x}) = \arg\max_i \boldsymbol{w}_i \cdot \boldsymbol{x} + b_i.$$

We will often write this as

$$f(\boldsymbol{x}) = \arg\max_i f_i(\boldsymbol{x}),$$

where each f_i is of the form

$$f_i(\boldsymbol{x}) = \boldsymbol{w}_i \cdot \boldsymbol{x} + b_i.$$

The set of parameters of f is $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$. A popular form of linear classifier is *logistic regression* (Murphy 2012), where the model has the form

$$f(\boldsymbol{x}) = \arg\max_i y_i(\boldsymbol{x}),$$
$$y_i(\boldsymbol{x}) = \frac{\exp f_i(\boldsymbol{x})}{\sum_j \exp f_j(\boldsymbol{x})}.$$

Note that this model has the following properties:

- $0 \leq y_i(\boldsymbol{x}) \leq 1$ for any $\boldsymbol{x}$ and i;
- $\sum_i y_i(\boldsymbol{x}) = 1$.

As such, the vector $\boldsymbol{y}(\boldsymbol{x})$ can be interpreted as a probability mass function over the different classes of $\boldsymbol{x}$, and f simply assigns $\boldsymbol{x}$ to the class that is deemed most likely.

2.4.2 Multi-Layer Perceptron

Perhaps the simplest possible neural network architecture is the Multi-Layer Perceptron (MLP, Goodfellow et al. 2016; Murphy 2012), also called a Feedforward Network. In general, an MLP is a series of logistic regression models stacked on top of each other. The final layer can be either a logistic or a linear regression model, depending on whether the MLP is used to solve a classification or a regression problem. In this thesis, we will restrict our attention to the classification case. The general model of an MLP for multiclass classification has the form

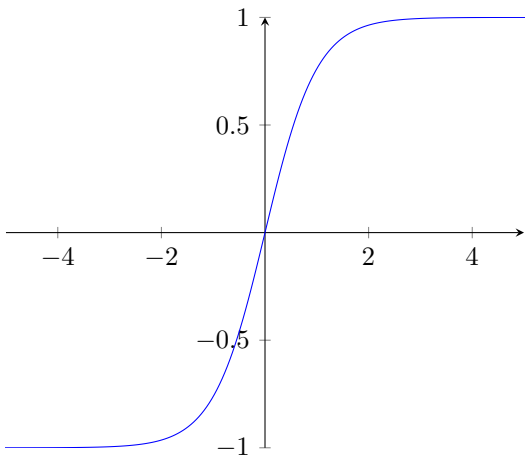
$$\begin{aligned}
\mathrm{P}(c \mid \boldsymbol{x}, \theta) &= \frac{\exp(\boldsymbol{w}_c \cdot \boldsymbol{h}_L(\boldsymbol{x}) + b_c)}{\sum_{c'=1}^{C} \exp(\boldsymbol{w}_{c'} \cdot \boldsymbol{h}_L(\boldsymbol{x}) + b_{c'})}, \\
\boldsymbol{h}_L(\boldsymbol{x}) &= g_L(\boldsymbol{V}_L \boldsymbol{h}_{L-1}(\boldsymbol{x}) + \boldsymbol{b}_L), \\
&\vdots \\
\boldsymbol{h}_2(\boldsymbol{x}) &= g_2(\boldsymbol{V}_2 \boldsymbol{h}_1(\boldsymbol{x}) + \boldsymbol{b}_2), \\
\boldsymbol{h}_1(\boldsymbol{x}) &= g_1(\boldsymbol{V}_1 \boldsymbol{x} + \boldsymbol{b}_1).
\end{aligned}$$

The functions $\boldsymbol{h}_i$ are called *hidden layers*[1], $\mathrm{P}(c \mid \boldsymbol{x}, \theta)$ is called the *output layer*, the matrices $\boldsymbol{V}_l$ are called the *weights*, the vectors $\boldsymbol{b}_l$ and $\boldsymbol{b}$ are the *biases* and the g_i are non-linear *activation functions*, applied component-wise to their input vectors (see Figure 2.2 for an illustration of a simple MLP). In the past, the common choice for these functions was the logistic sigmoid, hyperbolic tangent (Goodfellow et al. 2016; Murphy 2012) or softplus (Dugas et al. 2001). However, more recently, other activation functions such as the Rectified Linear Unit (ReLU, Dahl et al. 2013; Glorot et al. 2011) and Maxout (Ian J. Goodfellow et al. 2013) combined with Dropout (Srivastava et al. 2014) have obtained better results (see Figure 2.1 for examples of some activation functions). In general, any useful activation function will be non-linear, piecewise differentiable and monotonic. The set of parameters to be optimized by the learning algorithm is given by $\theta = \{\boldsymbol{V}_1, \ldots, \boldsymbol{V}_L, \boldsymbol{W}, \boldsymbol{B}\}$ where $\boldsymbol{W} = (\boldsymbol{w}_1 \mid \cdots \mid \boldsymbol{w}_C)$ and $\boldsymbol{B} = (\boldsymbol{b}_1 \mid \cdots \mid \boldsymbol{b}_L \mid \boldsymbol{b})$. The classification rule f associated with such a model is the same as the linear classifiers: we simply compute $\boldsymbol{w}_c \cdot \boldsymbol{h}_L(\boldsymbol{x})$ for every class c and assign $\boldsymbol{x}$ to the class that maximizes this quantity. Thus we define

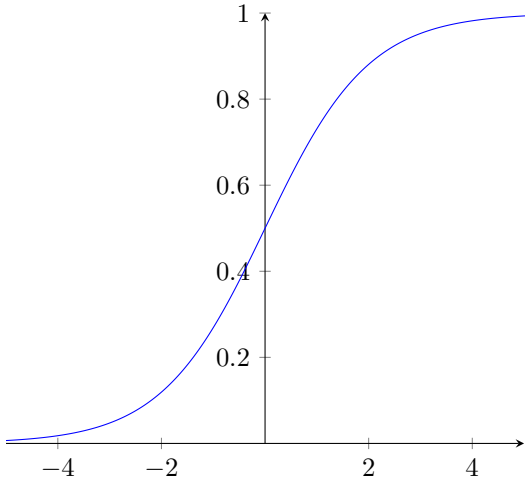
$$f(\boldsymbol{x}) = \arg\max_c \boldsymbol{w}_c \cdot \boldsymbol{h}_L(\boldsymbol{x}) + b_c. \tag{2.4}$$

Moreover, we will assume that none of the matrices $\boldsymbol{V}_i$ are equal to zero. This will be necessary for the theorems in later chapters to work, but it is a realistic assumption: if any $\boldsymbol{V}_i$ were equal to zero, the MLP would essentially discard its entire input. This is never done in practice.

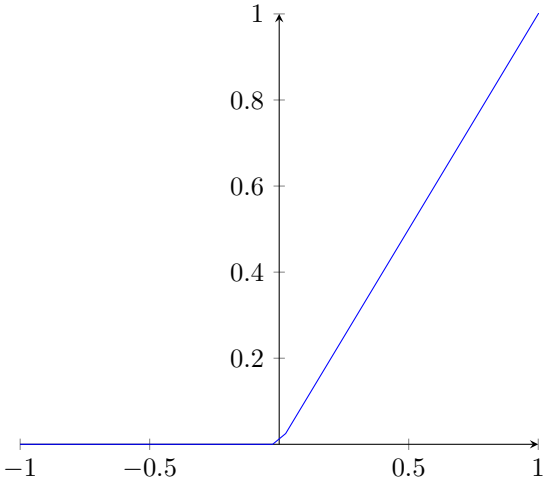
[1] For notational convenience, we will assume the input layer is given by the function $\boldsymbol{h}_0(\boldsymbol{x}) = \boldsymbol{x}$.



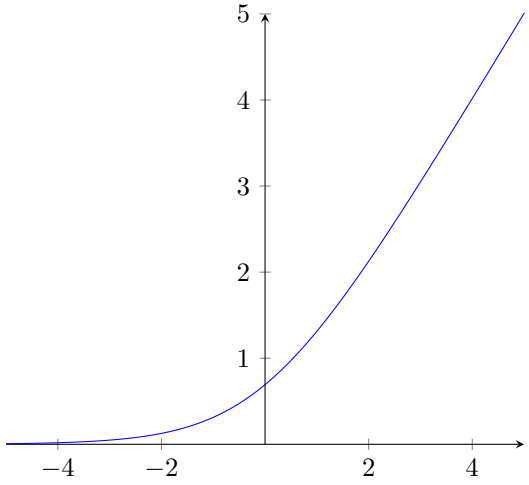
(a) The hyperbolic tangent



(b) The logistic sigmoid



(c) The rectified linear unit (ReLU)



(d) Softplus

Figure 2.1: Some examples of commonly used activation functions.

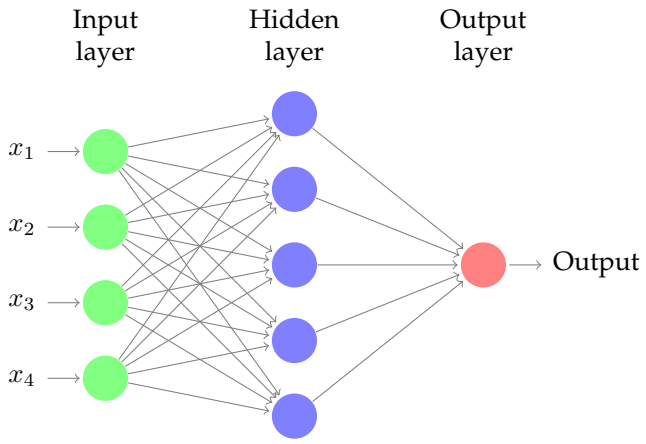


Figure 2.2: Graphical depiction of an MLP with four inputs, one hidden layer and a single output value. Each connection between an input x_i and a hidden node h_j has an associated weight v_{ij}, given by a matrix $\boldsymbol{V}$. The connections between the hidden nodes and the output node are also weighted by a matrix $\boldsymbol{W}$, which in this particular case is just a single vector. Such an MLP could be used for binary classification, where the output can be interpreted as the probability of the input belonging to class 0 (or class 1, depending on the training procedure).

The power of the MLP lies in the fact that, under very general conditions, it is a universal function approximator (Hornik 1991): an MLP with one hidden layer using the sigmoid activation function is capable of modelling any sufficiently smooth function within any desired level of accuracy. The level of accuracy depends only on the number of hidden units used: the more hidden units the MLP has, the better it will be able to model its target function. This so-called Universal Approximation Theorem (UAT) also holds for other activation functions, such as ReLU and Maxout. Thus, for any given function f, the question is never *if* an MLP can be used to model it, but rather *how*. For even though the UAT states that a single hidden layer is sufficient, it is known (Bengio 2009) that as the number of layers of an MLP decreases, the number of hidden units required to model any given function can increase exponentially. The question of how to find the most suitable architecture of an MLP for a given problem is generally a challenging problem.

2.4.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are deep neural networks mainly intended for image recognition tasks, as they are inspired by the visual cortex of the brain (Min et al. 2016). The visual cortex consists of a hierarchy of two cell types: simple cells and complex cells (Hubel et al. 1968). The simple cells are sensitive to primitive patterns in subregions of the visual field, whereas the complex cells synthesize the information from simple cells to identify more intricate patterns. CNNs implement four key ideas that take advantage of the properties of natural signals: local connectivity, shared weights, pooling and the use of many layers (Lecun et al. 2015).

The typical structure of a CNN, as introduced in LeCun et al. 1998a, is shown in Figure 2.3. The first few stages are composed of two types of layers: convolutional and pooling layers. Each unit in a convolutional layer receives input from a set of units located in a small neighborhood of the previous layer. This *local connectivity* allows neurons to extract elementary visual features such as oriented edges and corners. Since these elementary feature detectors are likely to be useful across the entire image instead of just a local neighborhood, units in a convolutional layer are organized in planes within which all units share the same weights. The set of outputs of units in such a plane is called a *feature map*. These outputs are then passed through a non-linearity, typically a ReLU. When the CNN is used for classification, the signals are passed through a pooling layer in order to reduce the sensitivity of the network to all sorts of distortions. After repeating this process of convolution, non-linearity and pooling a few times, the results may additionally

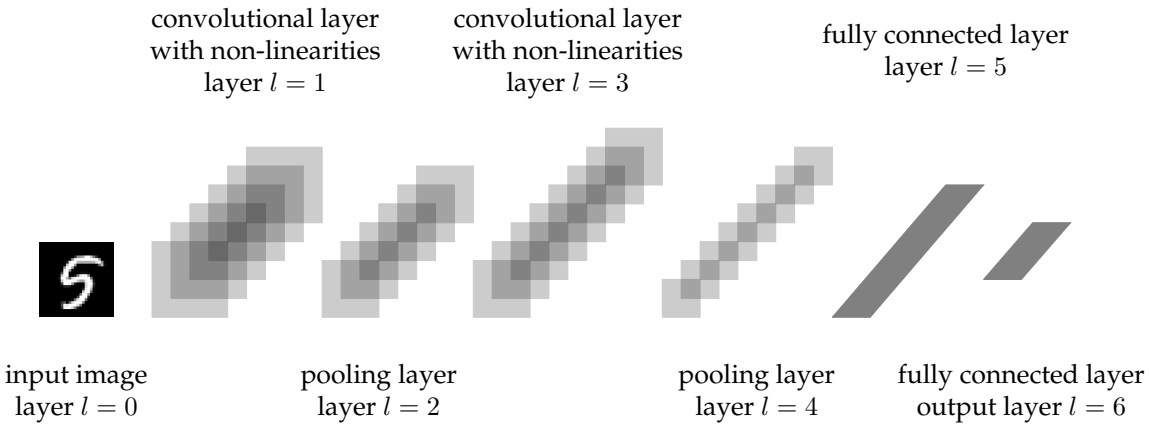


Figure 2.3: A typical CNN architecture.

be passed through fully-connected layers like a regular MLP.

Mathematical description

One of the most important operations in a CNN is the *convolution*, to which they owe their name. In practice, the convolutional layers of a CNN implement a discrete two-dimensional convolution as follows. Let $\mathbf{X} \in \mathbb{R}^{c \times d \times d}$ be the input to the convolutional layer with activation function g (usually ReLU or some variation of it, see Gu et al. 2015) and let $\mathbf{W} \in \mathbb{R}^{c \times q \times q}$ be its *filter* (also called *kernel*) with *bias* $b \in \mathbb{R}$. The feature map of this layer is then given by

$$g(\mathbf{W} \star \mathbf{X} + b), \tag{2.5}$$

where

$$(\mathbf{W} \star \mathbf{X})_{ij} = \sum_{l=1}^{c} \sum_{m=1}^{q} \sum_{n=1}^{q} w_{lmn} x_{l,m+s(i-1),n+s(j-1)} \tag{2.6}$$

for all $i, j = 1, \ldots, \frac{d-q}{s} + 1$. Here, the s parameter is called the *stride*. In case the stride is implicitly clear, we will leave it out; sometimes, however, we will use the notation $\boldsymbol{W} \star_s \boldsymbol{X}$ to refer explicitly to a convolution of $\mathbf{W}$ and $\mathbf{X}$ using stride s. The result of such an operation yields a tensor of size $c \times t \times t$ where $t = \frac{d-q}{s} + 1$. It is important to note that q and s cannot be arbitrary integers: s must be a divisor of $d - q$ and we must have $q \leq d$, otherwise the operation makes no sense.

The discrete convolution operation outlined above can actually be viewed as a matrix multiplication where the matrix is constrained to have some entries equal to others: each row is required to be equal to the row above it shifted by one element. In two dimensions, such matrices are called *doubly block circulant matrices*. Moreover, because the kernels are usually much smaller than the input, these matrices will be sparse (i.e. have many entries equal to zero). Hence, from a purely theoretical perspective, neural networks employing convolution operations are no different from regular MLPs.

It is common for a convolutional layer to have many different feature maps, which are "stacked" on top of eachother to produce a 3D output tensor. However, as is evident from the equations, each feature map has one set of weights which it applies across the entire input.

After each convolution operation, usually a pooling layer is applied. On input a matrix $\boldsymbol{X} \in \mathbb{R}^{d \times d}$, output (i, j) of a pooling layer of size $q \times q$ is given by

$$p(\{x_{n+s(i-1),m+s(j-1)} \mid 1 \leq n, m \leq q\}), \tag{2.7}$$

where p is the pooling operation, s is the stride and $i, j = 1, \ldots, \frac{d-q}{s} + 1$. The pooling operation is usually either $\max$ or an L_p norm, though many alternatives exist (Gu et al. 2015). In case the input to the pooling layer is a 3D tensor rather than a 2D matrix, the pooling operation is simply applied to each matrix independently along the third axis. The result is another 3D tensor consisting of downsampled matrices.

After several repetitions of convolution and pooling layers, the result is a 3D tensor which can be passed into a regular MLP for further processing. Before this can happen, of course, the tensor is flattened into a one-dimensional vector. Mathematically, the operation of flattening is trivial as it is merely a change in notation. An $m \times n$ matrix $\boldsymbol{X} = [\boldsymbol{x}_1 \mid \cdots \mid \boldsymbol{x}_n]$ is flattened into an mn vector which we will write as $\overline{\boldsymbol{X}}$. This vector is given by the relation

$$x_{ij} = \overline{X}_{n(i-1)+j}$$

for all $1 \leq i \leq m$ and $1 \leq j \leq n$. That is, the flattened vector $\overline{\boldsymbol{X}}$ can be constructed from the matrix $\boldsymbol{X}$ by simply concatenating all rows of $\boldsymbol{X}$ into one vector. The transformation $f(i, j)$ of the original indices into the indices of the flattened vector is given by

$$f(i, j) = n(i - 1) + j.$$

Conversely, the inverse transformation is given by

$$f_1(k) = \mathrm{quo}(k - 1, n) + 1,$$
$$f_2(k) = \mathrm{rem}(k - 1, n) + 1,$$

where $\mathrm{rem}(a, b)$ is the remainder when a is divided by b and $\mathrm{quo}(a, b)$ is the quotient of a divided by b. It is easily seen that $f_1(f(i, j)) = i$ and $f_2(f(i, j)) = j$. Furthermore, the flattening operation is easily extended to higher-dimensional tensors $\mathbf{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$.

2.5 Generating adversarial perturbations

An important question is how we are going to estimate actual minimal adversarial perturbations for an input to a given classifier. This will be important later, when lower bounds on the robustness of classifiers are put to the test. For complex classifiers, solving this problem exactly would require the solution of a prohibitively expensive optimization problem[2]. Fortunately, several efficient methods have been proposed in the literature to solve this exact problem:

- Szegedy et al. 2013 approximate adversarial perturbations using a box-constrained L-BFGS (Byrd et al. 1995);
- Goodfellow et al. 2015 proposed a method called *fast gradient sign*;
- Rozsa et al. 2016a introduce a hot/cold approach for adversarial example generation, as well as *fast gradient value*.

We discuss each method in turn.

2.5.1 L-BFGS

Let c be some class label. The aim of the L-BFGS method is to minimize $\|\boldsymbol{r}\|$ subject to $f(\boldsymbol{x} + \boldsymbol{r}) = c$ and $\boldsymbol{x} + \boldsymbol{r} \in [0, 1]^d$. Informally, $\boldsymbol{x} + \boldsymbol{r}$ is the input closest to $\boldsymbol{x}$ classified as c by f. Obviously, the task of finding such an $\boldsymbol{r}$ is non-trivial only when $f(\boldsymbol{x}) \neq c$. However, since the exact computation of this perturbation is a hard problem in general (Murty et al. 1987), it is approximated using a box-constrained L-BFGS. Szegedy et al. 2013 use line-search to find the minimum $\alpha > 0$ such that the minimizer $\boldsymbol{r}$ of the following optimization problem satisfies $f(\boldsymbol{x} + \boldsymbol{r}) = c$:

[2]Indeed, this optimization problem will be non-convex in general, and so solving it will be NP-complete (Murty et al. 1987).

Minimize $\alpha \|\boldsymbol{r}\| + \mathcal{L}(\boldsymbol{x} + \boldsymbol{r}, c)$ subject to $\boldsymbol{x} + \boldsymbol{r} \in [0, 1]^d$.

The constraint that $\boldsymbol{x} + \boldsymbol{r} \in [0, 1]^d$ ensures that the adversarial example has the same range of pixel values as the original data, and that it lies within the domain of the original function. One may omit this constraint for simplicity, however, because the perturbations $\boldsymbol{\eta}$ are typically small and thus do not move the input too far. This method is extremely effective, finds very small perturbations, and can cause the model to output specific, desired classes, and makes no assumptions about the structure of the model, but is also highly expensive, requiring multiple calls to an iterative optimization procedure for each example (Warde-Farley et al. 2016).

2.5.2 Fast gradient sign

The fast gradient sign method simplifies the problem of finding adversarial perturbations by fixing the allowed size of the perturbation and maximizing the cost incurred by the perturbation (Warde-Farley et al. 2016):

Maximize $\mathcal{L}(\boldsymbol{x} + \boldsymbol{\eta}, \theta)$ subject to $\|\boldsymbol{\eta}\|_\infty \leq \varepsilon$.

Here, ε is a hyperparameter chosen by the user. To obtain a fast, closed-form solution to this optimization problem, Goodfellow et al. 2015 use a first-order Taylor approximation of the loss function, yielding

Maximize $\mathcal{L}(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \theta) \cdot \boldsymbol{\eta}$ subject to $\|\boldsymbol{\eta}\|_\infty \leq \varepsilon$.

The solution is given by

$$\boldsymbol{\eta} = \varepsilon \operatorname{sign}(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \theta)). \tag{2.8}$$

In principle, one can use higher-order approximations of the loss function to obtain higher-order methods for generating adversarial perturbations. An issue with this method, though, is that it computes adversarial perturbations which are not necessarily minimal. In order to find perturbations whose Euclidean norms are as small as possible, we need to choose a value for ε such that $\boldsymbol{\eta}$ is an adversarial perturbation yet $\|\boldsymbol{\eta}\|$ is minimal. Clearly $\|\boldsymbol{\eta}\|$ is proportional to $|\varepsilon|$ so we need to find a value for ε that is as close to zero as possible. This can be accomplished by a simple binary search as shown in Algorithm 2.1. The value of ε found by the algorithm will lie within the interval $[0, M]$, where M depends upon the data set in question. It is important that M be set small enough so that any natural sample that is perturbed by M in any direction should not change classification[3]. Otherwise, the classifier is not wrong in changing its classification when such a perturbation is applied.

2.5.3 Hot/cold

The hot/cold approach by Rozsa et al. 2016a is inspired by the technique of *image inverting* by Mahendran et al. 2015, designed to reconstruct an image which minimizes the loss for a given class represented by a one-hot feature vector in the penultimate layer (the last hidden layer right before the softmax is applied). In the hot/cold model, Rozsa et al. 2016a use the output from the penultimate layer to define directions in the input space that help move toward a target class, the *hot class*, while moving away from the original class, the *cold class*. This is done by computing a hot/cold feature vector $\boldsymbol{\omega}$ for an input $\boldsymbol{x}$ as follows. Let c be the cold class, i.e. $f(\boldsymbol{x}) = c$, the original class of the input $\boldsymbol{x}$. Define a hot class $c' \neq c$ and let

$$\omega_j(\boldsymbol{x}) = \begin{cases} |h_L^{(j)}(\boldsymbol{x})| & \text{if } j = c' \\ -h_L^{(j)}(\boldsymbol{x}) & \text{if } j = c \\ 0 & \text{otherwise} \end{cases}, \tag{2.9}$$

[3]The word "natural" is crucial here: for any $M > 0$, one can always choose a sample sufficiently close to a decision boundary such that a perturbation with norm less than M should change its classification. However, the natural samples (i.e. those that arise in practice) should rarely lie very close to a decision boundary.

Algorithm 2.1: Estimate the smallest possible ε for fast gradient sign

Data: Classifier f with parameters θ and loss function $\mathcal{L}$, input $\boldsymbol{x}$
Result: an estimated minimal value for ε
begin
  `/* compute the gradient signs */`
  $\boldsymbol{s} \leftarrow \text{sign}(\nabla_{\boldsymbol{x}} \mathcal{L}(\theta, \boldsymbol{x}, f(\boldsymbol{x})))$
  `/* binary search for the smallest` ε `*/`
  $u \leftarrow 1$
  $l \leftarrow 0$
  $\varepsilon \leftarrow \frac{u}{2}$
  while $u - l > 0$ **do**
    if $f(\boldsymbol{x}) = f(\boldsymbol{x} + \varepsilon \boldsymbol{s})$ **then**
      $l \leftarrow \varepsilon$
    else
      $u \leftarrow \varepsilon$
    end
    $\varepsilon \leftarrow l + \frac{u-l}{2}$
  end
  return ε
end

where $\boldsymbol{h}_L$ is the output of the penultimate layer and $h_L^{(i)}$ denotes its ith component. Finally, backpropagation is used to estimate the changes needed to the input in order to move according to the constructed feature vector $\boldsymbol{\omega}$. A clear advantage of this technique is that it can generate multiple perturbations for the same input, whereas other techniques such as L-BFGS, FGS and FGV can only generate one perturbation for every input. However, as Rozsa et al. 2016a stress, the resulting perturbations are not minimal. This is intentional, as the goal of the hot/cold approach is to use the resulting perturbations to improve the robustness of neural networks. Rozsa et al. 2016a argue this can be done more effectively using *hard positives* formed by non-minimal perturbations for data augmentation.

2.5.4 Fast gradient value

Besides their hot/cold approach, Rozsa et al. 2016a also propose a method they call *fast gradient value*. This is essentially the same as fast gradient sign, only they use the full gradient of the loss function instead of just the signs. The estimated adversarial perturbation is thus determined by

$$\boldsymbol{\eta} = \varepsilon \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y, \theta), \tag{2.10}$$

where ε is again a parameter of the algorithm. Minimizing this parameter using a slightly modified version of Algorithm 2.1 should yield estimates for minimal adversarial perturbations.

2.5.5 Evaluation

Based on efficiency considerations, only the FGS method with binary search is used in this work to generate adversarial perturbations. It is significantly faster than the L-BFGS method and can yield very small perturbations using a simple algorithm.

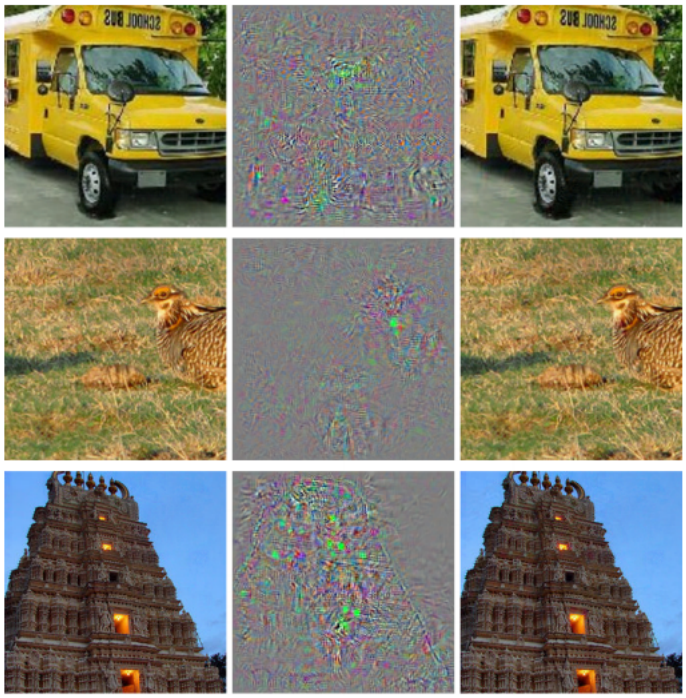


Figure 2.4: Example of adversarial perturbations from Szegedy et al. 2013. The left column shows the original, correctly classified images; the center column shows the perturbation; the right column shows the resulting image, which is consistently classified as "ostrich".

2.6 Protections against adversarial perturbations

When one wishes to devise protective measures against adversarial examples, it is first necessary to stress that such examples will always exist in any classifier that does not obtain 100% accuracy[4]. Hence, it is a futile undertaking to attempt to eliminate this phenomenon completely, as classifiers trained on real-world data sets rarely achieve 100% accuracy. The problem with adversarial examples is therefore not so much the mere fact that they exist, but the fact that they are much more abundant than would be expected given the accuracy of state-of-the-art classifiers. Indeed, one would expect that if a sample lies so close to a decision boundary its classification can be changed by an imperceptibly small perturbation, that either this sample looks like nothing in particular or it clearly looks like some sort of interpolation between both classes. As it turns out, this is not the case at all in practice: adversarial examples look like natural samples from a completely different class than the one the classifier assigns them, even if that classifier has high accuracy. Figure 1.1 provides a striking example of this, but there are many more. Figure 2.4 shows another set of examples of adversarial perturbations taken from Szegedy et al. 2013. The cross-model and cross-dataset generalization capabilities of these examples (Goodfellow et al. 2015), as well as the discovery of universal adversarial perturbations (Moosavi-Dezfooli et al. 2016) only add more to the mystery. The mysterious nature of adversarial examples has not stopped researchers from speculating and proposing methods for defending against them, however. In this section, we discuss several of those proposed defensive mechanisms.

[4] And even if it does, in practice such accuracy is achieved on a test set, not the entire space of possible inputs. So a classifier achieving 100% test set accuracy may still perform poorly on some samples.

2.6.1 Deep Contractive Network

The Deep Contractive Network (DCN, Gu et al. 2014) is based on the idea that if we want a neural network to be robust to small perturbations, then the components of its Jacobian matrix should be small. Ideally, such a model should then be regularized by adding the following term to the loss function:

$$\lambda \sum_{i=1}^{N} \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}_i} \right\|. \tag{2.11}$$

Even for relatively small networks, however, this penalty is computationally expensive to compute. Therefore, the following simplification is penalized instead:

$$\sum_{i=1}^{N} \sum_{j=1}^{L} \lambda_j \left\| \frac{\partial \boldsymbol{h}_j(\boldsymbol{x}_i)}{\partial \boldsymbol{h}_{j-1}(\boldsymbol{x}_i)} \right\|.$$

This is a layer-wise contractive penalty which can be computed much more efficiently than the full penalty of Equation (2.11). However, it limits the capacity of the neural network and does not guarantee optimality of Equation (2.11).

2.6.2 Generative Adversarial Networks

The framework of Generative Adversarial Networks (GANs, Goodfellow et al. 2014) estimates a generative model G for the data distribution μ by simultaneously training two different models: the generator G and a discriminator D. These models have competing objectives: the goal of D is to learn to tell whether a sample came from the real distribution μ or whether it was generated by G; the goal of G is to maximize the probability of D making a mistake. It can be shown that the globally optimal solution to this problem occurs when G fully captures the training data distribution and D outputs 50% on every sample.

GANs could be useful to protect against adversarial examples (Xia et al. 2016). The generator G is trained to fool the discriminator D, so obviously adversarial examples provide a weakness that G can exploit in order to produce natural images from μ which fool D. As such, D can be trained to resist these examples and become robust to such perturbations. By alternately training both models, G can find more sophisticated adversarial examples while D can become increasingly robust against more advanced attacks.

2.6.3 Fast Gradient Sign

Goodfellow et al. 2015 use their FGS method as a regularizer by optimizing the following objective:

$$\mathcal{L}'(\boldsymbol{x}, \theta) = \alpha \mathcal{L}(\boldsymbol{x}, \theta) + (1 - \alpha) \mathcal{L}(\boldsymbol{x} + \varepsilon \operatorname{sign} \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \theta), \theta).$$

In their experiments, they set $\alpha = 0.5$. They found this to be an effective regularizer, reducing the error rate on adversarial examples by 10%.

2.6.4 Foveations

Lou et al. 2016 use a foveation-based mechanism to defend CNNs against adversarial examples. They define a foveation as a transformation of the image that selects a region in which the CNN is applied, discarding the information from the other regions. Foveations always have the same size, even if the size of the selected region varies. To achieve this, linear interpolations are applied to resize the images. The idea behind this approach is based on several hypotheses. First, Lou et al. 2016 assume that CNNs act linearly only on some regions in the input, but non-linearly in others. Secondly, the authors assume that the accuracy of a CNN should not be significantly affected by their foveation transformations, as CNNs should be invariant to changes in scale and position. Finally, they assume the robustness of CNNs to changes in scale and position does not generalize to adversarial perturbations. Their results suggest that foveations are indeed a powerful approach to increasing robustness to adversarial perturbations.

2.6.5 Dimensionality reduction

Maharaj 2016 propose to reduce the dimensionality of the input to achieve robustness to adversarial perturbations. Their idea is based on the observation that of the 256^3 different colors supported by the RGB format, humans can only perceive about half of these. Hence, it makes little sense to train CNNs to distinguish between pixel colors which appear identical to human eyes. Discretizing pixel intensities thus appears to be a viable approach to not only decrease the dimensionality of the problem space (and make it easier to train the models), but also achieve robustness to adversarial perturbations: because only discrete values of intensities are permitted, there is a minimum allowed distortion to each image before it can change.

2.6.6 Stability training

Zheng et al. 2016 introduce stability training, where a regularizing term called a stability objective is added to the loss function:

$$\Omega(\boldsymbol{x}, \boldsymbol{x}', \theta) = d(f(\boldsymbol{x}), f(\boldsymbol{x}')).$$

Here, $\boldsymbol{x}$ is the original input to the network, $\boldsymbol{x}'$ is a perturbed copy of $\boldsymbol{x}$ and d is a suitable metric on the feature space of the network. The idea is that the perturbed copy $\boldsymbol{x}'$ should lie so close to the original $\boldsymbol{x}$ in the input space that the outputs $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ of the classifier should lie close to each other as well. In order to implement this sort of training, it is necessary to generate small perturbed samples of each input at each iteration of training.

2.6.7 Batch-Adjusted Network Gradients

Rozsa et al. 2016b propose a method called Batch-Adjusted Network Gradients (BANG) based on their hypothesis of evolutionary stalling. The intuition behind evolutionary stalling is that, during training, the learning algorithm only optimizes the model just enough so that it correctly classifies each sample in the training set. As a result, many samples will lie close to the decision boundary and a small perturbation is sufficient to cause them to change class. Rozsa et al. 2016b resolve this problem by making sure the gradients of every training sample do not vanish during training, by normalizing them in such a way that even correctly classified samples continue to contribute significantly to the weight updates. Specifically, instead of a single learning rate parameter η as is normally used in gradient descent, they use a separate learning rate for each layer in the network and each sample in the batch.

2.6.8 Hot/cold

Rozsa et al. 2016a use their hot/cold approach to generate adversarial examples for data augmentation. This approach has the advantage over FGS in that it can produce multiple different perturbations for the same sample, whereas FGS can produce only one. Rozsa et al. 2016a explicitly use non-minimal perturbations which they call "hard positives" in order to ensure robustness of the network.

2.6.9 Defensive distillation

Defensive distillation (Papernot et al. 2016) modifies the training procedure as follows. First, a "teacher" network is trained using standard techniques. This teacher network is evaluated on every training sample to produce *soft labels*. These soft labels usually assign non-zero probability to classes the input sample does not belong to. This is in contrast to the typical hard labels used in standard training, where 100% probability is assigned to the correct class and 0% to all other classes. Finally, a "distilled" network is trained using the soft labels produced by the teacher network instead of the typical hard labels. By training on the soft labels, the model should overfit the data less and try to be more regular. However, according to Carlini et al. 2016, this method only protects against a specific form of adversarial examples and leaves the distilled network vulnerable to other classes of perturbations.

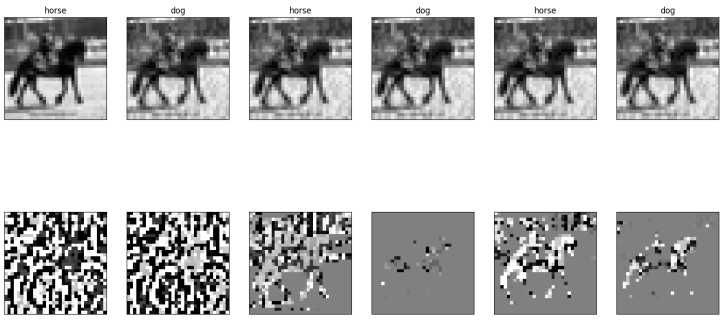


Figure 2.5: Iterative perturbation

2.7 Iterative perturbation

A technique which may shed more light on the nature of adversarial perturbations, is to take a random sample $\mathbf{X}$ from the dataset $\mathcal{D}$ and iteratively apply the FGS method to adversarially perturb it. This creates a sequence of images $\mathbf{X}_0, \ldots, \mathbf{X}_n$ where $\mathbf{X}_0 = \mathbf{X}$ and $\mathbf{X}_{i+1}$ is an adversarial perturbation of $\mathbf{X}_i$. Figure 2.5 illustrates the result of this technique on a CNN for $n = 5$. The figure has two rows: the top row displays the resulting images; the bottom row displays the perturbation applied to the previous image to obtain the image above it (the first perturbation being the difference between the first and last image). This technique has the interesting effect of producing perturbations which are themselves recognizable images. From the third image onwards (i.e. the perturbation of the second image), it appears that the perturbations resulting in a classification of "horse" actually resemble horses, and perturbations resulting in "dog" resemble dogs. It is also clear from the images that the norm of the perturbations diminishes dramatically after the first image. This can be explained as follows. The first adversarial perturbation projects the input image onto the closest decision boundary. Subsequent adversarial perturbations then merely have to shift the image slightly in the right direction in order to cross the boundary again into the original class. This also explains why the labels alternate between two classes: the FGS method simply "zig-zags" back and forth across the decision boundary of the same two classes.

These findings appear to support the evolutionary stalling hypothesis of Rozsa et al. 2016b as they show that the classifier has not yet captured all the relevant features of the different classes in order to properly separate them. Hence, it seems plausible that the learning simply "stalls" as more and more samples are classified correctly.

Chapter 3

Linear classifiers

The first family of classifiers to be studied in this thesis is the family of linear classifiers. Linear classifiers are among the simplest classifiers used in practice (e.g. for logistic regression). Many neural network classifiers can in fact be seen as "nothing more" than linear classifiers of their feature spaces (although these feature spaces are usually highly non-linear transformations of the input spaces). Therefore, deriving lower bounds on the robustness of linear classifiers will enable us to derive lower bounds on the robustness of the output layer of neural networks later on. Thus, this family provides an important starting point for our analysis.

3.1 Binary classification

In the binary case, an important simplification can be made. If f is a binary linear classifier, then it is characterized by two functions $f_0(\boldsymbol{x}) = \boldsymbol{w}_0 \cdot \boldsymbol{x} + b_0$ and $f_1(\boldsymbol{x}) = \boldsymbol{w}_1 \cdot \boldsymbol{x} + b_1$. The classification rule then boils down to

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } f_1(\boldsymbol{x}) \geq f_0(\boldsymbol{x}) \\ 0 & \text{if } f_1(\boldsymbol{x}) < f_0(\boldsymbol{x}) \end{cases} . \tag{3.1}$$

Note that $f_1(\boldsymbol{x}) \geq f_0(\boldsymbol{x})$ if and only if $(\boldsymbol{w}_1 - \boldsymbol{w}_0) \cdot \boldsymbol{x} + b_1 - b_0 \geq 0$, so f can equivalently be characterized by a single function $g(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$ where $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_0$, $b = b_1 - b_0$ and the following rule:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } g(\boldsymbol{x}) \geq 0 \\ 0 & \text{if } g(\boldsymbol{x}) < 0 \end{cases} . \tag{3.2}$$

For ease of notation, we will always characterize a binary linear classifier as in Equation (3.2).

We write $\Delta_{\text{adv}}(\boldsymbol{x}; f \mid f(\boldsymbol{x}) = 1)$ and $\Delta_{\text{adv}}(\boldsymbol{x}; f \mid f(\boldsymbol{x}) = 0)$ for the norm of the minimal perturbation in case $\boldsymbol{x}$ is such that $f(\boldsymbol{x}) = 1$ and $f(\boldsymbol{x}) = 0$, respectively:

$$\Delta_{\text{adv}}(\boldsymbol{x}; f \mid f(\boldsymbol{x}) = 1) = \min_{\boldsymbol{r} \in \mathbb{R}^d} \{\|\boldsymbol{r}\| \mid f(\boldsymbol{x} + \boldsymbol{r}) = 0\} = \min_{\boldsymbol{r} \in \mathbb{R}^d} \{\|\boldsymbol{r}\| \mid \boldsymbol{w} \cdot (\boldsymbol{x} + \boldsymbol{r}) + b \leq 0\}, \tag{3.3}$$

$$\Delta_{\text{adv}}(\boldsymbol{x}; f \mid f(\boldsymbol{x}) = 0) = \min_{\boldsymbol{r} \in \mathbb{R}^d} \{\|\boldsymbol{r}\| \mid f(\boldsymbol{x} + \boldsymbol{r}) = 1\} = \min_{\boldsymbol{r} \in \mathbb{R}^d} \{\|\boldsymbol{r}\| \mid \boldsymbol{w} \cdot (\boldsymbol{x} + \boldsymbol{r}) + b > 0\}. \tag{3.4}$$

Obviously

$$\Delta_{\text{adv}}(\boldsymbol{x}; f) = \begin{cases} \Delta_{\text{adv}}(\boldsymbol{x}; f \mid f(\boldsymbol{x}) = 0), & \text{if } f(\boldsymbol{x}) = 0 \\ \Delta_{\text{adv}}(\boldsymbol{x}; f \mid f(\boldsymbol{x}) = 1), & \text{if } f(\boldsymbol{x}) = 1 \end{cases} . \tag{3.5}$$

The decision boundary is defined by the set of all perturbations $\boldsymbol{r}$ such that $\boldsymbol{w} \cdot (\boldsymbol{x} + \boldsymbol{r}) + b = 0$. These cause the classifier to assign equal probability to both classes, and any further perturbation one way or another will cross the decision boundary into one of the two classes. It is therefore interesting to attempt to

determine analytically which perturbations $\boldsymbol{r}$ satisfy $\boldsymbol{w} \cdot (\boldsymbol{x} + \boldsymbol{r}) + b = 0$ while simultaneously minimizing $\|\boldsymbol{r}\|$. For this we can define a Lagrangian:

$$\mathcal{L}(\boldsymbol{r}, \lambda) = \|\boldsymbol{r}\|^2 + \lambda(\boldsymbol{w} \cdot (\boldsymbol{x} + \boldsymbol{r}) + b), \tag{3.6}$$

where we have used $\|\boldsymbol{r}\|^2$ for simplicity. Minimizing this leads to the system of equations

$$\begin{aligned} \forall j : \frac{\partial \mathcal{L}}{\partial r_j} &= 2r_j + \lambda w_j = 0, \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= \boldsymbol{w} \cdot (\boldsymbol{x} + \boldsymbol{r}) + b = 0. \end{aligned}$$

Therefore

$$r_j = -\frac{\lambda}{2} w_j.$$

We thus get

$$\begin{aligned} \sum_i w_i \left(x_i - \frac{\lambda w_i}{2} \right) + b &= 0 \\ 2 \sum_i w_i x_i + 2b &= \lambda \sum_i w_i^2. \end{aligned}$$

From this we conclude

$$\lambda = \frac{2(\boldsymbol{w} \cdot \boldsymbol{x} + b)}{\|\boldsymbol{w}\|^2}.$$

This yields the final expression for $\boldsymbol{r}$:

$$\boldsymbol{r} = -\frac{\boldsymbol{w} \cdot \boldsymbol{x} + b}{\|\boldsymbol{w}\|^2} \boldsymbol{w} \tag{3.7}$$

Perhaps unsurprisingly, when $b = 0$, $\boldsymbol{x} + \boldsymbol{r}$ is exactly the orthogonal projection of $\boldsymbol{x}$ onto the hyperplane spanned by $\boldsymbol{w}$ (see Figure 3.1). The Euclidean norm of this vector is

$$\|\boldsymbol{r}\| = \frac{|\boldsymbol{w} \cdot \boldsymbol{x} + b|}{\|\boldsymbol{w}\|}. \tag{3.8}$$

We can now prove the following

Lemma 3.1. *For a binary linear classifier f, the following equality holds for all $\boldsymbol{x}$:*

$$\Delta_{\text{adv}}(\boldsymbol{x}; f) = \frac{|\boldsymbol{w} \cdot \boldsymbol{x} + b|}{\|\boldsymbol{w}\|}.$$

Proof. Let $\boldsymbol{x}$ be a random sample from μ, let $\boldsymbol{r}$ be defined as in Equation (3.7) and let $\boldsymbol{q}$ be an adversarial perturbation for $\boldsymbol{x}$ (i.e. $f(\boldsymbol{x}) \neq f(\boldsymbol{x} + \boldsymbol{q})$). Assume for the sake of contradiction that $\|\boldsymbol{q}\| < \|\boldsymbol{r}\|$. Suppose first that $f(\boldsymbol{x}) = 0$, then $\boldsymbol{w} \cdot \boldsymbol{x} + b \leq 0$ and $\boldsymbol{w} \cdot (\boldsymbol{x} + \boldsymbol{q}) + b > 0$. It follows that $\boldsymbol{w} \cdot \boldsymbol{q} > 0$. The Cauchy-Schwarz inequality yields $|\boldsymbol{w} \cdot \boldsymbol{q}| \leq \|\boldsymbol{w}\| \, \|\boldsymbol{q}\| < \|\boldsymbol{w}\| \, \|\boldsymbol{r}\| = |\boldsymbol{w} \cdot \boldsymbol{x} + b|$. Thus $\boldsymbol{w} \cdot \boldsymbol{q} < -\boldsymbol{w} \cdot \boldsymbol{x} - b$ and so $\boldsymbol{w}(\boldsymbol{x} + \boldsymbol{q}) + b < 0$, which is a contradiction. The case where $f(\boldsymbol{x}) = 1$ can be handled analogously.

It follows that all adversarial perturbations $\boldsymbol{q}$ must satisfy $\|\boldsymbol{q}\| \geq \|\boldsymbol{r}\|$. However, it is easily seen by the definition of $\boldsymbol{r}$ that $\boldsymbol{r}$ itself is an adversarial perturbation. Thus it is a minimal adversarial perturbation and the lemma follows. □

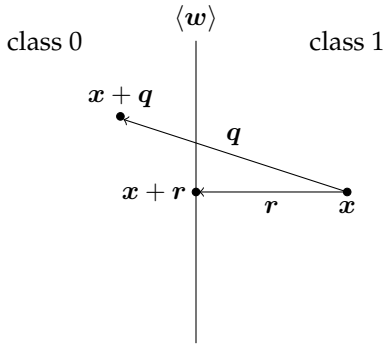


Figure 3.1: The minimal adversarial perturbation for an input $\boldsymbol{x}$ to a binary classifier is given by the orthogonal projection $\boldsymbol{x} + \boldsymbol{r}$ of $\boldsymbol{x}$ onto the hyperplane spanned by $\boldsymbol{w}$. Any other adversarial perturbation $\boldsymbol{q}$ must satisfy $\|\boldsymbol{q}\| \geq \|\boldsymbol{r}\|$.

Plugging this result into the definition of $\rho_{\text{adv}}(f)$, we find

Theorem 3.2. *For a binary classifier f, it holds that*

$$\rho_{\text{adv}}(f) = \frac{1}{\|\boldsymbol{w}\|} \mathrm{E}_{\mu}[|\boldsymbol{w} \cdot \boldsymbol{x} + b|].$$

Note that Theorem 3.2 suggests that the robustness of a binary linear classifier increases as $\|\boldsymbol{w}\|$ gets smaller. This may (at least partially) explain why weight decay works (Connect et al. 1992).

3.2 Multiclass classification

From the definition of a linear classifier it is easy to see that an observation $\boldsymbol{x}$ is classified as c if and only if $\boldsymbol{w}_c \cdot \boldsymbol{x} + b_c \geq \boldsymbol{w}_j \cdot \boldsymbol{x} + b_j$ for all $j \neq c$. For an observation $\boldsymbol{x}$ and any class $c \neq f(\boldsymbol{x})$, we can compute the norm of the smallest perturbation $\boldsymbol{r}$ such that $f(\boldsymbol{x} + \boldsymbol{r}) = c$. Let us denote this perturbation by $\Delta_{\text{adv}}(\boldsymbol{x}; f, c)$:

$$\Delta_{\text{adv}}(\boldsymbol{x}; f, c) = \min_{\boldsymbol{r} \in \mathbb{R}^d} \{\|\boldsymbol{r}\| \mid f(\boldsymbol{x} + \boldsymbol{r}) = c\}. \tag{3.9}$$

It is easily seen that

$$\Delta_{\text{adv}}(\boldsymbol{x}; f) = \min_{c \neq f(\boldsymbol{x})} \{\Delta_{\text{adv}}(\boldsymbol{x}; f, c)\}. \tag{3.10}$$

The restriction that $c \neq f(\boldsymbol{x})$ is important, since otherwise we would get a degenerate solution of $\Delta_{\text{adv}}(\boldsymbol{x}; f) = 0$ for all $\boldsymbol{x}$. Intuitively, in the multiclass case, $\Delta_{\text{adv}}(\boldsymbol{x}; f)$ is the norm of the projection of $\boldsymbol{x}$ onto the class "closest" to $\boldsymbol{x}$ but distinct from $f(\boldsymbol{x})$. A necessary condition for any adversarial perturbation $\boldsymbol{q}$ is given by the following

Lemma 3.3. *Let f be a linear classifier and let $\boldsymbol{q}$ be an adversarial perturbation for an instance $\boldsymbol{x}$ where $f(\boldsymbol{x}) = c$ and $f(\boldsymbol{x} + \boldsymbol{q}) = c' \neq c$. There exists an $\alpha \in (0, 1)$ such that $\boldsymbol{w}_c \cdot (\boldsymbol{x} + \alpha \boldsymbol{q}) + b_c = \boldsymbol{w}_{c'} \cdot (\boldsymbol{x} + \alpha \boldsymbol{q}) + b_{c'}$.*

Proof. A little algebra shows

$$\alpha = \frac{(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{x} + b_{c'} - b_c}{(\boldsymbol{w}_c - \boldsymbol{w}_{c'}) \cdot \boldsymbol{q}}. \tag{3.11}$$

It remains to be proven that $0 < \alpha < 1$. Note that, by assumption, $\boldsymbol{w}_c \cdot \boldsymbol{x} + b_c > \boldsymbol{w}_{c'} \cdot \boldsymbol{x} + b_{c'}$ and $\boldsymbol{w}_c \cdot (\boldsymbol{x} + \boldsymbol{q}) + b_c < \boldsymbol{w}_{c'} \cdot (\boldsymbol{x} + \boldsymbol{q}) + b_{c'}$. This implies $(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{x} + b_{c'} - b_c < 0$ and $(\boldsymbol{w}_c - \boldsymbol{w}_{c'}) \cdot \boldsymbol{q} < 0$, so $\alpha > 0$. Furthermore, since $\boldsymbol{w}_c \cdot (\boldsymbol{x} + \boldsymbol{q}) + b_c < \boldsymbol{w}_{c'} \cdot (\boldsymbol{x} + \boldsymbol{q}) + b_{c'}$, we find $\alpha < 1$. □

In other words, Lemma 3.3 states that for any linear classifier f, if we want to adversarially perturb an input $\boldsymbol{x}$ so its assigned label changes from c to c', we must cross the boundary where f assigns equal probability to those two classes. Note that this condition is only necessary, not sufficient: it is perfectly possible to cross this boundary for the classes c and c' at a point where some other class c'' is still more likely than either c or c', but we must cross this boundary nonetheless. Using Lemma 3.3 we find

Lemma 3.4. *Let f be a linear classifier and let $\boldsymbol{x}$ be any input such that $f(\boldsymbol{x}) = c'$. Then for all classes $c \neq c'$,*

$$\Delta_{\text{adv}}(\boldsymbol{x}; f, c) \geq \frac{|(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{x} + b_{c'} - b_c|}{\|\boldsymbol{w}_{c'} - \boldsymbol{w}_c\|}.$$

Proof. We can find the norm of the smallest perturbation $\boldsymbol{r}$ such that $\boldsymbol{w}_c \cdot (\boldsymbol{x} + \boldsymbol{r}) + b_c = \boldsymbol{w}_{c'} \cdot (\boldsymbol{x} + \boldsymbol{r}) + b_{c'}$ using the following Lagrangian:

$$\mathcal{L} = \|\boldsymbol{r}\|^2 + \lambda((\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot (\boldsymbol{x} + \boldsymbol{r}) + b_{c'} - b_c)$$

The solution to this optimization problem is given by

$$\boldsymbol{r} = \frac{(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{x} + b_{c'} - b_c}{\|\boldsymbol{w}_{c'} - \boldsymbol{w}_c\|^2}(\boldsymbol{w}_c - \boldsymbol{w}_{c'}).$$

Taking norms we get

$$\|\boldsymbol{r}\| = \frac{|(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{x} + b_{c'} - b_c|}{\|\boldsymbol{w}_{c'} - \boldsymbol{w}_c\|}.$$

By Lemma 3.3 and the construction of $\boldsymbol{r}$, any adversarial perturbation $\boldsymbol{q}$ must satisfy $\|\boldsymbol{q}\| > \|\boldsymbol{r}\|$. In particular, any adversarial perturbation $\boldsymbol{q}$ such that $f(\boldsymbol{x}+\boldsymbol{q}) = c$ satisfies $\|\boldsymbol{q}\| > \|\boldsymbol{r}\|$. Hence $\Delta_{\text{adv}}(\boldsymbol{x}; f, c) \geq \|\boldsymbol{r}\|$. □

The following lemma is a trivial consequence of Lemma 3.4:

Lemma 3.5. *Let f be a linear classifier. Then for all inputs $\boldsymbol{x}$ where $f(\boldsymbol{x}) = c$,*

$$\Delta_{\text{adv}}(\boldsymbol{x}; f) \geq \min_{c' \neq c} \frac{|(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{x} + b_{c'} - b_c|}{\|\boldsymbol{w}_{c'} - \boldsymbol{w}_c\|}.$$

Note how Lemma 3.5 confirms the intuition that the smallest adversarial perturbation to an instance $\boldsymbol{x}$ is bounded from below by the orthogonal projection of $\boldsymbol{x}$ onto the class closest to $\boldsymbol{x}$ but distinct from $f(\boldsymbol{x})$, since this is exactly the quantity on the right-hand side of the inequality. Finally, we get

Theorem 3.6. *Let f be a linear classifier. Then,*

$$\rho_{\text{adv}}(f) \geq \mathrm{E}_\mu \left[\min_{c' \neq c} \frac{|(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{x} + b_{c'} - b_c|}{\|\boldsymbol{w}_{c'} - \boldsymbol{w}_c\|} \right].$$

In the case of binary classification, we have $C = 2$, $\boldsymbol{w}_0 = \boldsymbol{w}_1 - \boldsymbol{w}$, $b = b_1 - b_0$, and $\boldsymbol{w}_1 = \boldsymbol{w} + \boldsymbol{w}_0$. Hence

$$\rho_{\text{adv}}(f) \geq \mathrm{E}_\mu \left[\frac{|2\boldsymbol{w} \cdot \boldsymbol{x} + b|}{\|2\boldsymbol{w}\|} \right] = \frac{1}{\|\boldsymbol{w}\|} \mathrm{E}_\mu[|\boldsymbol{w} \cdot \boldsymbol{x} + b|].$$

For ease of notation, we now introduce the following shorthand which will be used throughout the rest of this thesis:

$$\delta(\boldsymbol{x}; f) = \min_{c \neq f(\boldsymbol{x})} \frac{|(\boldsymbol{w}_c - \boldsymbol{w}_{f(\boldsymbol{x})}) \cdot \boldsymbol{x} + b_c - b_{f(\boldsymbol{x})}|}{\left\|\boldsymbol{w}_c - \boldsymbol{w}_{f(\boldsymbol{x})}\right\|}. \tag{3.12}$$

3.2.1 Computing the bound

In practice, of course, the distribution μ will be unknown and so we will not be able to compute expectations with respect to μ. Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \mid i = 1, \ldots, N\}$ of i.i.d. samples $\boldsymbol{x}_i$ from μ and class labels $y_i = f(\boldsymbol{x}_i)$ an empirical estimate of the lower bound may be computed using the sample mean instead of the expectation:

$$\hat{\rho}_{\text{adv}}(f) \geq \frac{1}{N} \sum_{i=1}^{N} \delta(\boldsymbol{x}_i; f). \tag{3.13}$$

3.2.2 Tightness of the bound

An interesting question to explore now is the tightness of this bound.

Theorem 3.7. *The bound of Theorem 3.6 is tight.*

Proof. Let f be a classifier for $C = d \geq 2$ classes where

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } x_1 > x_2, \ldots, x_C \\ 2 & \text{if } x_2 > x_1, x_3, \ldots, x_C \\ \vdots & \\ C & \text{if } x_C > x_1, \ldots, x_{C-1} \end{cases}.$$

This classifier is linear since it can be characterized by linear functions of the form

$$f_i(\boldsymbol{x}) = \boldsymbol{e}_i \cdot \boldsymbol{x} = x_i.$$

Hence, it is subject to the bound of Theorem 3.6, which in this case simplifies to

$$\Delta_{\text{adv}}(\boldsymbol{x}; f) \geq \min_{c' \neq c} |x_{c'} - x_c|.$$

It is easily seen, however, that this bound is exact for this particular classifier, i.e.

$$\Delta_{\text{adv}}(\boldsymbol{x}; f) = \min_{c' \neq c} |x_{c'} - x_c|.$$

The reason being that f classifies an input $\boldsymbol{x}$ into the class corresponding to the index of the maximal component of $\boldsymbol{x}$. Thus, $f(\boldsymbol{x}) = c$ if and only if x_c is the maximal component of $\boldsymbol{x}$. To find the norm of the minimal perturbation $\boldsymbol{r}$ such that $f(\boldsymbol{x} + \boldsymbol{r}) \neq c$, one simply takes $\boldsymbol{r} = (x_c - x_{c'})\boldsymbol{e}_{c'}$ where c' is the index of the second-highest component of $\boldsymbol{x}$. Clearly $f(\boldsymbol{x} + \boldsymbol{r}) = c' \neq c$ and $\|\boldsymbol{r}\| = |x_{c'} - x_c|$ is minimal. □

Theorem 3.7 shows that the bound from Theorem 3.6 is tight in the sense that there exist linear classifiers whose robustness is exactly equal to that bound.

3.3 Experiments

In order to test whether the theoretical bound is valid, several tests were performed on linear classifiers. The results are shown in Table 3.1. The different columns of the table are

- Classifiers: the number of classifiers on which this test was performed;
- Samples: the number of samples per classifier;
- Dimensionality: the number of components of the input vectors;

No.	Classifiers	Samples	Dimensionality	Classes	Sample interval	Parameter interval	Mean	Median	Std
1	100	10	2	5	$[-1, 1]$	$[-1, 1]$	0.9283	0.8637	0.5557648
2	100	10	20	5	$[-1, 1]$	$[-1, 1]$	3.783	3.782	0.5845007
3	100	10	2	50	$[-1, 1]$	$[-1, 1]$	1.015	0.9544	0.5454238
4	100	10	2	5	$[-10, 10]$	$[-1, 1]$	4.937	4.498	2.636395
5	100	10	2	5	$[-1, 1]$	$[-10, 10]$	1.061	1.076	0.5375943

Table 3.1: Test results on linear classifiers

- Classes: the number of output classes of the classifier;
- Sample interval: the interval from which the inputs were uniformly sampled;
- Parameter interval: the interval from which the classifier parameters were uniformly sampled;
- Mean: mean deviation of the actual bounds from the theoretical bound;
- Median: median bound deviation;
- Std: standard deviation of the bound deviations.

Figure 3.2 also shows boxplots of the bound deviations for each test. As Goodfellow et al. 2015 point out, the FGS method is exact in the case of linear classifiers, so the perturbations obtained using FGS in these tests were not approximations; they were the actual minimal adversarial perturbations.

3.3.1 Discussion

No bound violations were detected in any of the tests. We can see that tests 1, 3 and 5 yielded highly similar results, but 2 and 4 are clearly different. The second test increased the dimensionality of the samples by a factor of ten; as a result, the mean deviation increased approximately by a factor of three. The standard deviation remained about the same, however. The most significant change occurred in test 4, where the sample interval was enlarged from $[-1, 1]$ to $[-10, 10]$. The mean deviation and standard deviation increased almost by a factor of five and four, respectively. From these findings, we conclude that the theoretical bound is most sensitive to the dimensionality of the input as well as the range the input values can lie in.

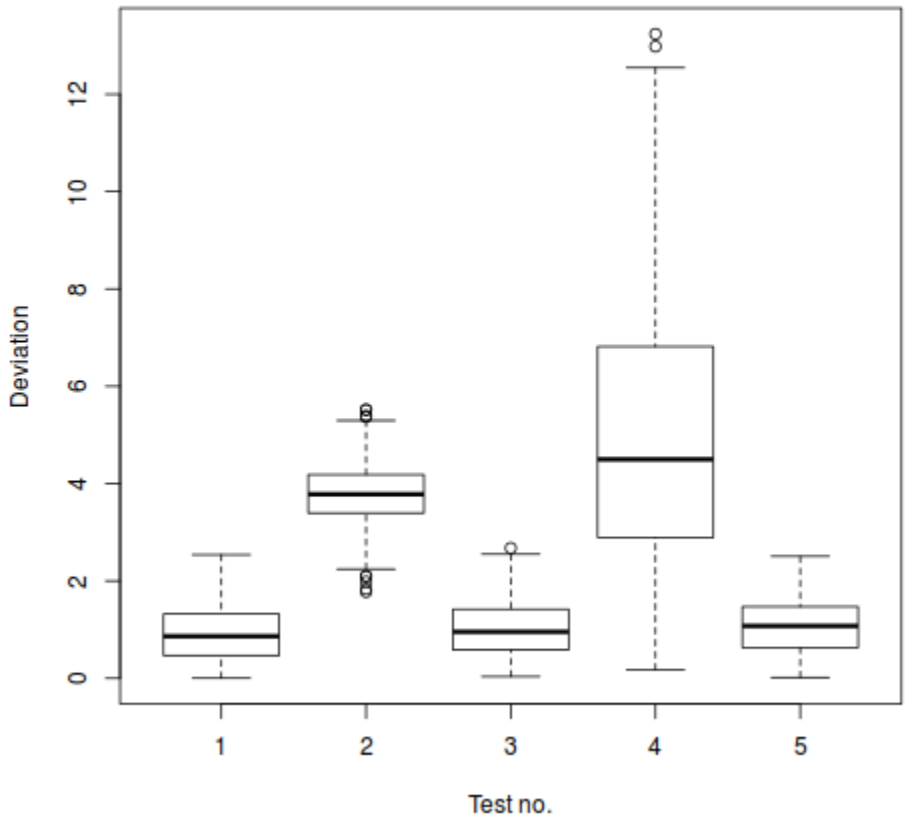


Figure 3.2: Boxplots of bound deviations for linear classifiers

Chapter 4

Multi-Layer Perceptrons

Multi-Layer Perceptrons are the most basic neural network models. An MLP used for classification usually takes its input, performs a series of linear transformations followed by non-linear activation functions and finally sends the result of these transformations to a linear classifier. Throughout this chapter we assume all MLPs end in a linear classifier of their feature space. This does not need to be the case in general, but this assumption allows us to use the bounds for linear classifiers in order to derive bounds on the robustness of MLPs.

4.1 Robustness

In Chapter 3 we already derived a tight lower bound for linear classifiers. Hence the robustness of MLPs to adversarial perturbations is already solved for $L = 0$, i.e. for MLPs consisting only of an input and output layer: the output layer of an MLP is a logistic regression model, which is at heart just a linear classifier. Here, we will attempt to extend the study to MLPs with arbitrarily many hidden layers. By Theorem 3.6 we know that the robustness of the output layer is bounded from below by

$$\mathrm{E}_{\mu}\left[\min_{c' \neq c} \frac{|(\boldsymbol{w}_{c'} - \boldsymbol{w}_c) \cdot \boldsymbol{h}_L(\boldsymbol{x}) + b_{c'} - b_c|}{\|\boldsymbol{w}_{c'} - \boldsymbol{w}_c\|}\right]. \tag{4.1}$$

However, such a result only applies for adversarial perturbations to $\boldsymbol{h}_L(\boldsymbol{x})$, not $\boldsymbol{x}$ itself. We need a procedure for transforming an adversarial perturbation $\boldsymbol{q}$ to $\boldsymbol{h}_L(\boldsymbol{x})$ into an adversarial perturbation $\boldsymbol{r}$ to $\boldsymbol{x}$ such that

$$\boldsymbol{h}_L(\boldsymbol{x}) + \boldsymbol{q} = \boldsymbol{h}_L(\boldsymbol{x} + \boldsymbol{r}). \tag{4.2}$$

Equation (4.2) only has solutions if the perturbation $\boldsymbol{q}$ is such that $\boldsymbol{h}_L(\boldsymbol{x}) + \boldsymbol{q}$ is still within the range of $\boldsymbol{h}_L$. This is a realistic constraint which we will enforce when attempting to solve this problem, because in practice, we only get to perturb the original input $\boldsymbol{x}$, not any of the outputs of the intermediate layers. Hence it is safe to assume $\boldsymbol{h}_L(\boldsymbol{x}) + \boldsymbol{q}$ will always lie within the range of $\boldsymbol{h}_L$.

4.1.1 One-layer MLP

For clarity, the model of an MLP with one hidden layer is given by

$$\begin{aligned} \mathrm{P}(c \mid \boldsymbol{x}, \theta) &= \frac{\exp(\boldsymbol{w}_c \cdot \boldsymbol{h}(\boldsymbol{x}) + b'_c)}{\sum_{c'=1}^{C} \exp(\boldsymbol{w}_{c'} \cdot \boldsymbol{h}(\boldsymbol{x}) + b'_{c'})}, \\ \boldsymbol{h}(\boldsymbol{x}) &= g(\boldsymbol{V}\boldsymbol{x} + \boldsymbol{b}). \end{aligned}$$

Equation (4.2) in this context becomes

$$\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{q} = \boldsymbol{h}(\boldsymbol{x} + \boldsymbol{r}). \tag{4.3}$$

Unfortunately, unless $\boldsymbol{V}$ and g are invertible, this problem has no analytical solution. It will sometimes be the case that g is invertible, but $\boldsymbol{V}$ will almost never be a square matrix and hence almost never invertible. However, we know that $\|\boldsymbol{q}\|$ is lower bounded by Equation (4.1), so perhaps it is possible to utilize that fact along with Equation (4.3) to derive a lower bound for $\|\boldsymbol{r}\|$.

In the remainder we shall assume $\boldsymbol{h} : \mathbb{R}^d \to \mathbb{R}^n$ satisfies the requirements of Theorem A.1. We may then write

$$\boldsymbol{h}(\boldsymbol{x} + \boldsymbol{r}) = \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\boldsymbol{r} + \boldsymbol{\varepsilon}, \tag{4.4}$$

where $\boldsymbol{\varepsilon}$ is the vector of error terms satisfying

$$\|\boldsymbol{\varepsilon}\| \leq \frac{M}{2}\sqrt{n}\,\|\boldsymbol{r}\|^2. \tag{4.5}$$

We thus obtain

$$\boldsymbol{q} = \boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\boldsymbol{r} + \boldsymbol{\varepsilon}. \tag{4.6}$$

Taking norms, this yields

$$\|\boldsymbol{q}\| \leq \|\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\|\,\|\boldsymbol{r}\| + \|\boldsymbol{\varepsilon}\|, \tag{4.7}$$

where $\|\boldsymbol{J}_{\boldsymbol{h}}\|$ is the induced Euclidean matrix norm of $\boldsymbol{J}_{\boldsymbol{h}}$. Note the inequality $\|\boldsymbol{A}\boldsymbol{x}\| \leq \|\boldsymbol{A}\|\,\|\boldsymbol{x}\|$ holds for these norms, a property we used to derive Equation (4.7). Plugging in the upper bound on $\|\boldsymbol{\varepsilon}\|$, we obtain a quadratic inequality in $\|\boldsymbol{r}\|$:

$$\|\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\|\,\|\boldsymbol{r}\| + \frac{M}{2}\sqrt{n}\,\|\boldsymbol{r}\|^2 - \|\boldsymbol{q}\| \geq 0$$

Its only viable solution is given by

$$\|\boldsymbol{r}\| \geq \frac{\sqrt{\|\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\|^2 + 2M\sqrt{n}\,\|\boldsymbol{q}\|} - \|\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\|}{M\sqrt{n}}. \tag{4.8}$$

Based on these findings, we can formulate the following theorem:

Theorem 4.1. *Let f be an MLP classifier with one hidden layer and assume $\boldsymbol{h} : \mathbb{R}^d \to \mathbb{R}^n$ satisfies the requirements of Theorem A.1. Then,*

$$\Delta_{\text{adv}}(\boldsymbol{x}; f) \geq \frac{\sqrt{\|\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\|^2 + 2M\sqrt{n}\delta(\boldsymbol{h}(\boldsymbol{x}); f)} - \|\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\|}{M\sqrt{n}}.$$

Note that for this result to work, we have to assume additionally that $M \neq 0$. This presents no problem in practice, because if $M = 0$ then $\boldsymbol{h}$ is in fact a linear function which would contradict the assumption that g is non-linear.

4.1.2 L-layer MLP

We are now ready to state and prove the general robustness bound for MLPs with an arbitrary number of hidden layers:

Theorem 4.2. *Let f be an MLP classifier with L hidden layers where $\boldsymbol{h}_L : \mathbb{R}^d \to \mathbb{R}^n$ satisfies the requirements of Theorem A.1. Then for any $\boldsymbol{x} \in \mathbb{R}^d$,*

$$\Delta_{\text{adv}}(\boldsymbol{x}; f) \geq \frac{\sqrt{\|\boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\|^2 + 2M\sqrt{n}\delta(\boldsymbol{h}_L(\boldsymbol{x}); f)} - \|\boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\|}{M\sqrt{n}}.$$

Proof. Applying Theorem A.1 to $\boldsymbol{h}_L$ we obtain

$$\boldsymbol{h}_L(\boldsymbol{x} + \boldsymbol{r}) = \boldsymbol{h}_L(\boldsymbol{x}) + \boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\boldsymbol{r} + \boldsymbol{\varepsilon},$$

where

$$\|\boldsymbol{\varepsilon}\| \leq \frac{M}{2}\sqrt{n}\,\|\boldsymbol{r}\|^2.$$

Thus $\boldsymbol{q} = \boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\boldsymbol{r} + \boldsymbol{\varepsilon}$ and

$$\|\boldsymbol{q}\| \leq \|\boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\|\,\|\boldsymbol{r}\| + \frac{M}{2}\sqrt{n}\,\|\boldsymbol{r}\|^2.$$

This is a quadratic inequality in $\|\boldsymbol{r}\|$ whose solution is given by

$$\|\boldsymbol{r}\| \geq \frac{\sqrt{\|\boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\|^2 + 2M\sqrt{n}\,\|\boldsymbol{q}\|} - \|\boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\|}{M\sqrt{n}}.$$

By Theorem 3.6 we know

$$\|\boldsymbol{q}\| \geq \min_{c' \neq c} \frac{|(\boldsymbol{w}_c - \boldsymbol{w}_{c'}) \cdot \boldsymbol{h}_L(\boldsymbol{x})|}{\|\boldsymbol{w}_c - \boldsymbol{w}_{c'}\|}.$$

Hence the theorem follows. □

As before, we may assume $M \neq 0$. An immediate consequence of Theorem 4.2 is

Theorem 4.3. *Let f be an MLP classifier with L hidden layers where $\boldsymbol{h}_L : \mathbb{R}^d \to \mathbb{R}^n$ satisfies the requirements of Theorem A.1. Then,*

$$\rho_{\text{adv}}(f) \geq \mathrm{E}_\mu\left[\frac{\sqrt{\|\boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\|^2 + 2M\sqrt{n}\delta(\boldsymbol{h}_L(\boldsymbol{x}); f)} - \|\boldsymbol{J}_{\boldsymbol{h}_L}(\boldsymbol{x})\|}{M\sqrt{n}}\right].$$

Again, this quantity will usually not be computable since the distribution μ will be unknown. However, as before, we can approximate the bound using the sample mean.

4.2 Realism of the assumptions

One might rightly ask how realistic the result from Theorem 4.3 actually is. After all, we needed to assume that the function $\boldsymbol{h}_L$ satisfied the list of assumptions from Theorem A.1. In this section, I will try to show that these assumptions are quite realistic by deriving them from other realistic assumptions on the activation functions. Specifically, we have the following

Theorem 4.4. *Let f be an MLP classifier with L hidden layers whose activation functions g_i are twice differentiable on $\mathbb{R}$. Assume also that for each g_i there exist $N_i > 0$ and $M_i > 0$ such that $|g_i'(x)| \leq N_i$ and $|g_i''(x)| \leq M_i$ for all x. Then $\boldsymbol{h}_L(\boldsymbol{x}) = [h_L^{(1)}(\boldsymbol{x}), \ldots, h_L^{(n)}(\boldsymbol{x})]^T$ satisfies the following properties:*

1. *each* $h_L^{(i)}$ *is twice differentiable;*
2. $|\partial^\alpha h_L^{(i)}(\boldsymbol{x})| \leq M$ *for all* i, $|\alpha| = 2$ *and some* $M > 0$*;*
3. $|\partial^\alpha h_L^{(i)}(\boldsymbol{x})| \leq N$ *for all* i, $|\alpha| = 1$ *and some* $N > 0$*.*

Proof. The proof proceeds by induction on L. The case where $L = 0$ is trivial, so suppose

1. each $h_L^{(i)}$ is twice differentiable;
2. $|\partial^\alpha h_L^{(i)}(\boldsymbol{x})| \leq M'$ for all i, $|\alpha| = 2$ and some $M' > 0$;
3. $|\partial^\alpha h_L^{(i)}(\boldsymbol{x})| \leq N'$ for all i, $|\alpha| = 1$ and some $N' > 0$.

We need to show that $\boldsymbol{h}_{L+1} : \mathbb{R}^d \to \mathbb{R}^n : \boldsymbol{x} \mapsto [h_{L+1}^{(1)}(\boldsymbol{x}), \ldots, h_{L+1}^{(n)}(\boldsymbol{x})]^T$ (where d is the dimensionality of the input) then satisfies the following properties:

1. $h_{L+1}^{(i)}$ is twice differentiable for all i;
2. there exists an $M > 0$ such that $|\partial^\alpha h_{L+1}^{(i)}(\boldsymbol{x})| \leq M$ for all $\boldsymbol{x}$, $|\alpha| = 2$ and i;
3. there exists an $N > 0$ such that $|\partial^\alpha h_{L+1}^{(i)}(\boldsymbol{x})| \leq N$ for all $\boldsymbol{x}$, $|\alpha| = 1$ and i.

Since $\boldsymbol{h}_{L+1}(\boldsymbol{x}) = g_{L+1}(\boldsymbol{V}_{L+1}\boldsymbol{h}_L(\boldsymbol{x}) + \boldsymbol{b}_{L+1})$ we find

$$h_{L+1}^{(i)}(\boldsymbol{x}) = g_{L+1}\left(\sum_j v_{L+1,i,j} h_L^{(j)}(\boldsymbol{x}) + b_{L+1,i}\right).$$

Clearly, since g_{L+1} is twice differentiable by assumption and $h_L^{(j)}$ is twice differentiable for all j by the induction hypothesis, $h_{L+1}^{(i)}$ is twice differentiable for all i. This shows Item 1. To show Item 2, we distinguish two cases. First, let

$$\alpha_j = \begin{cases} 2 & j = k \\ 0 & \text{otherwise} \end{cases}$$

for some $k \in \{1, \ldots, d\}$. Then

$$\begin{aligned}
\partial^\alpha h_{L+1}^{(i)} = \frac{\partial^2 h_{L+1}^{(i)}}{\partial x_k^2} &= \frac{\partial}{\partial x_k}\left(g'_{L+1}\left(\sum_j v_{L+1,i,j} h_L^{(j)}(\boldsymbol{x}) + b_{L+1,i}\right)\sum_j v_{L+1,i,j}\frac{\partial}{\partial x_k} h_L^{(j)}(\boldsymbol{x})\right) \\
&= g''_{L+1}\left(\sum_j v_{L+1,i,j} h_L^{(j)}(\boldsymbol{x}) + b_{L+1,i}\right)\left(\sum_j v_{L+1,i,j}\frac{\partial}{\partial x_k} h_L^{(j)}(\boldsymbol{x})\right)^2 + \\
&\quad g'_{L+1}\left(\sum_j v_{L+1,i,j} h_L^{(j)}(\boldsymbol{x}) + b_{L+1,i}\right)\sum_j v_{L+1,i,j}\frac{\partial^2}{\partial x_k^2} h_L^{(j)}(\boldsymbol{x}).
\end{aligned}$$

Taking absolute values and applying the induction hypothesis, this yields

$$|\partial^\alpha h_{L+1}^{(i)}| \leq M_{L+1}\left(N'\sum_j |v_{L+1,i,j}|\right)^2 + N_{L+1}M'\sum_j |v_{L+1,i,j}|.$$

Hence we may choose

$$M = M_{L+1} \left(N's\right)^2 + N_{L+1} M' s,$$

where

$$s = \max_i \sum_j |v_{L+1,i,j}|.$$

For the second case, let

$$\alpha_j = \begin{cases} 1 & j \in \{k_1, k_2\} \\ 0 & \text{otherwise} \end{cases}$$

for $k_1, k_2 \in \{1, \ldots, d\}$ and $k_1 < k_2$. Of course, this case only applies when $d > 1$. We find

$$\begin{aligned} \partial^\alpha h_{L+1}^{(i)} &= \frac{\partial^2 h_{L+1}^{(i)}}{\partial x_{k_1} \partial x_{k_2}} = \frac{\partial}{\partial x_{k_2}} \left(g'_{L+1} \left(\sum_j v_{L+1,i,j} h_L^{(j)}(\boldsymbol{x}) + b_{L+1,i} \right) \sum_j v_{L+1,i,j} \frac{\partial}{\partial x_{k_1}} h_L^{(j)}(\boldsymbol{x}) \right) \\ &= g''_{L+1} \left(\sum_j v_{L+1,i,j} h_L^{(j)}(\boldsymbol{x}) + b_{L+1,i} \right) \sum_j v_{L+1,i,j} \frac{\partial}{\partial x_{k_2}} h_L^{(j)}(\boldsymbol{x}) \sum_j v_{L+1,i,j} \frac{\partial}{\partial x_{k_1}} h_L^{(j)}(\boldsymbol{x}) \\ &+ g'_{L+1} \left(\sum_j v_{L+1,i,j} h_L^{(j)}(\boldsymbol{x}) + b_{L+1,i} \right) \sum_j v_{L+1,i,j} \frac{\partial^2}{\partial x_{k_1} \partial x_{k_2}} h_L^{(j)}(\boldsymbol{x}) \end{aligned}$$

Again, taking absolute values and applying the induction hypothesis:

$$|\partial^\alpha h_{L+1}^{(i)}(\boldsymbol{x})| \leq M_{L+1} \left(N' \sum_j |v_{L+1,i,j}| \right)^2 + N_{L+1} M' \sum_j |v_{L+1,i,j}|$$

The result is identical to the first case.

Finally, to show Item 3, we let

$$\alpha_j = \begin{cases} 1 & j = k \\ 0 & \text{otherwise} \end{cases}$$

for some $k \in \{1, \ldots, d\}$. Then

$$\partial^\alpha h_{L+1}^{(i)} = \frac{\partial h_{L+1}^{(i)}}{\partial x_k} = g'_{L+1} \left(\sum_j v_{L+1,i,j} h_L^{(j)}(\boldsymbol{x}) + b_{L+1,i} \right) \sum_j v_{L+1,i,j} \frac{\partial}{\partial x_k} h_L^{(j)}(\boldsymbol{x}).$$

Again taking absolute values and applying the induction hypothesis, we find

$$|\partial^\alpha h_{L+1}^{(i)}| \leq N_{L+1} N' \sum_j |v_{L+1,i,j}|.$$

Hence we may choose

$$N = (N_{L+1} N') \max_i \sum_j |v_{L+1,i,j}|.$$

This completes the proof. □

By Theorem 4.4, in order for Theorem 4.2 to hold it is sufficient that the activation functions of the MLP in question be twice differentiable and have bounded first and second derivatives. These assumptions are not unrealistic: they are satisfied by the logistic sigmoid function, for example. The logistic sigmoid is in fact just a scaled and shifted version of the hyperbolic tangent:

$$\operatorname{sigm}(x) = \frac{1}{2}\tanh\left(\frac{x}{2}\right) + \frac{1}{2}. \tag{4.9}$$

Since $\tanh$ is twice differentiable, so is sigm. Moreover, the first and second derivatives of $\tanh$ are bounded by 1, so the first and second derivatives of sigm are bounded as well. In fact, $|\tanh(x)| \leq 1$ for all x.

The ReLU activation function presents some problems, however, as it is not differentiable at zero. Gradient-based optimization requires all activation functions be differentiable, though, so in practice either a smooth approximation to ReLU is used which is differentiable, such as the softplus function $\ln(1+\exp(x))$, or the value of the derivative is simply set to zero at the origin (Glorot et al. 2011). In both cases it can be seen that ReLU (as it is used in practice) also satisfies the necessary assumptions for Theorem 4.2 to hold.

4.3 Experiments

Practically speaking, we are interested in two different questions:

1. Is the theoretical bound of Theorem 4.2 ever violated?
2. How much can an actual adversarial perturbation deviate from this bound?

To answer the first question, we evaluate the bound on a number of randomly generated MLPs and check if the norms of the adversarial perturbations ever violate it. The basic methodology is as follows for a given dataset $\mathcal{D} = \{\boldsymbol{x}_i \mid i = 1, \ldots, N\}$:

1. Randomly generate an MLP f.
2. For each sample $\boldsymbol{x}_i \in \mathcal{D}$:
   (a) Estimate a minimal adversarial perturbation $\boldsymbol{q}$ to $\boldsymbol{x}_i$ for f.
   (b) Compute the theoretical lower bound δ of Theorem 4.2 for $\boldsymbol{x}_i$ and f.
   (c) If $\delta > \|\boldsymbol{q}\|$, signal an error.
3. Signal that no violations of the theoretical bound occured.

The above process can then be iterated several times to ensure the results are statistically significant. Note that we do not need any actual class labels attached to the samples $\boldsymbol{x}_i$: all we care about in this scenario is how much we need to perturb a given sample in order to make the MLP misclassify it. Hence, the MLP does not need to be trained; all we need are the class labels it assigns to the samples. This also raises the question where the samples are going to come from. We can parametrize the dimensionality d of the vectors $\boldsymbol{x}_i$ and simply generate N uniform random samples from the real vector space $[a, b]^d$ where $a, b \in \mathbb{R}$.

For the second question, we also keep track of the deviations during the randomized tests. However, since randomly generated MLPs are most likely not representative of actual trained MLPs, we also train a few MLPs for real classification tasks. We then also check if the bound is ever violated and how much the bound can deviate from the norms of actual adversarial perturbations.

4.3.1 M parameter

The computation of the theoretical bound is straightforward except for the value of M. It is important that this value be computed as precisely as possible: if we overestimate M, the bound becomes less tight; if we underestimate M, the bound may be violated. Fortunately, the proof of Theorem 4.4 provides a

	sigm	tanh	softplus	ReLU
First-order	$\frac{1}{4}$	1	1	1
Second-order	$\frac{5-3\sqrt{3}}{30\sqrt{3}-54}$	$\frac{\sinh\left(\frac{1}{2}\ln(2+\sqrt{3})\right)}{\cosh\left(\frac{1}{2}\ln(2+\sqrt{3})\right)^3}$	$\frac{1}{4}$	0

Table 4.1: Overview of bounds for first and second derivatives of some activation functions

way to estimate M in a layer-wise fashion. Algorithm 4.1 shows how this is done. Note, though, that the values A_i and B_i still need to be manually determined. This is easy to do for most commonly used activation functions; see Table 4.1. It is easily seen from Algorithm 4.1 that M could be made equal to zero for networks containing ReLU activation functions, because the second derivative is bounded by zero. This would violate the necessary assumption that $M > 0$, causing the computation of the theoretical bound to fail. There are several options to work around this problem:

1. initialize M_0 to a small positive value instead of 0;
2. return $\max(\varepsilon, M_L)$ at the end instead of M_L, where $\varepsilon > 0$ is small;
3. set the bound on the second-order derivative of ReLU to a small positive constant instead of 0.

After trying each of these possibilities, the one that seems to work best is the third option, where we set the bound of the second-order derivative of ReLU to 1 instead of 0. Note that Algorithm 4.1 will not, in general, compute an exact value for M; instead, the result will be an upper bound. From a theoretical point of view, this is fine since the value computed by Algorithm 4.1 still satisfies the necessary properties; practically speaking, however, this will make the bound less tight. The main reason why we use the estimate of Algorithm 4.1 instead of a more exact value is simple: Algorithm 4.1 runs in $\mathcal{O}(LV_RV_C)$ time where L is the number of layers and V_R and V_C are the maximum number of rows and columns, respectively, of the weight matrices of the network. For large networks, this is much more efficient than attempting to directly maximize $|\partial^\alpha h_L^{(i)}(\boldsymbol{x})|$ with respect to $\boldsymbol{x}$ for all i and $|\alpha| = 2$.

Algorithm 4.1: Computation of M

Data: MLP f with L hidden layers, activation functions g_i satisfying $|g_i'(x)| \leq A_i$ and $|g_i''(x)| \leq B_i$ for all x.
Result: a value M satisfying $|\partial^\alpha h_L^{(i)}(\boldsymbol{x})| \leq M$ for all $\boldsymbol{x}$, $|\alpha| = 2$ and i
begin
  $M_0 \leftarrow 0$
  $N_0 \leftarrow 1$
  for *i from 1 to L* **do**
    $s \leftarrow \max_j \sum_k |v_{i,j,k}|$
    $M_i \leftarrow B_i N_{i-1}^2 s^2 + A_i M_{i-1} s$
    $N_i \leftarrow A_i N_{i-1} s$
  end
  return M_L
end

4.3.2 Results

Randomized tests

The results are shown in Table 4.2. The different columns are

No.	Samples	Dimensionality	MLPs	Layers	Units	Classes	Sample interval	Parameter interval	Mean	Median	Std
1	10	2	100	3	10	5	$[-1, 1]$	$[-1, 1]$	0.6322801	0.5546744	0.3358796
2	10	2	100	3	10	5	$[-100, 100]$	$[-1, 1]$	0.6404704	0.5849356	0.4163827
3	10	2	100	3	10	5	$[-1, 1]$	$[-100, 100]$	0.4335774	0.3039111	0.4211190
4	10	20	100	3	10	5	$[-1, 1]$	$[-1, 1]$	1.1928828	0.9919565	0.9066600
5	10	2	100	3	30	5	$[-1, 1]$	$[-1, 1]$	0.6533218	0.6304373	0.3057876
6	10	2	100	3	10	50	$[-1, 1]$	$[-1, 1]$	0.6276122	0.6120808	0.2672927
7	10	2	100	6	20	5	$[-1, 1]$	$[-1, 1]$	0.6630859	0.6050979	0.2996120
8	100	2	10	3	10	5	$[-10, 10]$	$[-1, 1]$	0.6561273	0.6170216	0.3212896

Table 4.2: Test results on random MLPs

- Samples: the number of samples used in this test.
- Dimensionality: the number of components per sample.
- MLPs: the number of MLPs tested with these parameters.
- Layers: number of layers per MLP.
- Units: number of hidden units per MLP.
- Classes: number of output classes for each MLP.
- Sample interval: the dataset was generated by sampling all components uniformly from this interval.
- Parameter interval: the parameters of the MLPs were generated by sampling them uniformly from this interval.
- Mean: average deviation from the theoretical bound.
- Median: median deviation from the theoretical bound.
- Std: Standard deviation of the deviations from the theoretical bound.

Figure 4.1 shows boxplots of average bound deviations for each MLP.

Tests on trained MLPs

Iris dataset The Iris dataset, obtained from the UCI Machine Learning Repository (Lichman 2013), is a classic dataset in the field of statistics. The dataset consists of four continuous attributes, and each item in the dataset can belong to one of three classes. The dataset contains fifty examples for each class for a total of 150 samples. The MLP we trained on this dataset consisted of one hidden layer with five hidden units and a $\tanh$ activation function. The dataset was first standardized and then split up into 70% training data, 20% test data and 10% validation data. The resulting MLP achieves an accuracy of 100% on its test set.

Wine dataset The Wine dataset (Lichman 2013) consists of thirteen continuous attributes. The items can be classified into one of three classes, each representing a type of wine. The different classes have 59, 71 and 48 examples, respectively, for a total of 178 samples. These were standardized and randomly partitioned into 70% training data, 20% test data and 10% validation data. The MLP used has two hidden layers with five and three hidden units, respectively, and both utilize $\tanh$ activation functions. The MLP achieved an accuracy of 100% on its test set.

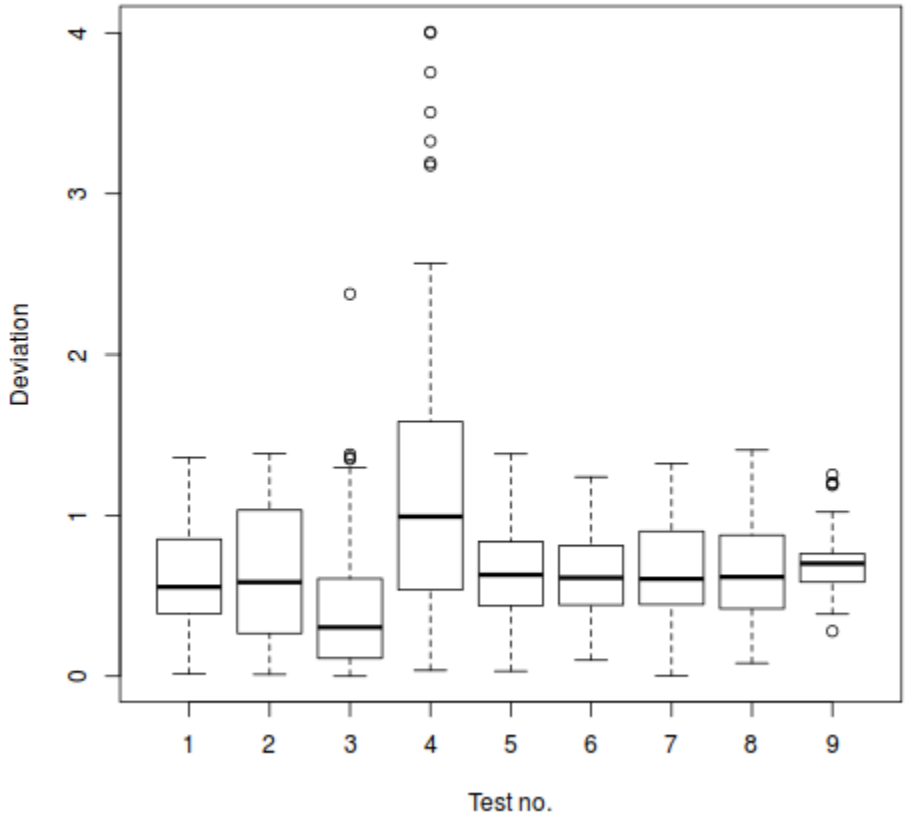


Figure 4.1: Boxplots of average bound deviations for MLP tests

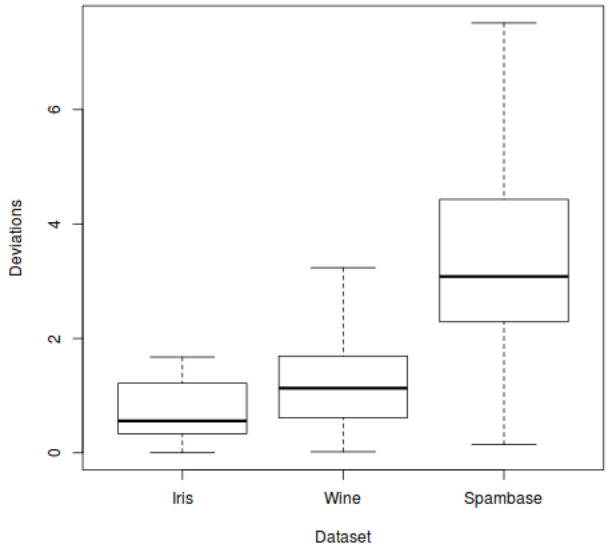


Figure 4.2: Boxplots of deviations of adversarial perturbations from the theoretical bound in trained MLPs

	Mean	Median	Std	Min	Max
Iris	0.715300	0.557900	0.4791929	0.002236	1.675000
Wine	1.26300	1.13100	0.775758	0.01861	3.23700
Spambase	3.4130	3.0830	1.499317	0.1446	7.5140

Table 4.3: Trained MLP statistics of bound deviations

Spambase dataset The Spambase dataset (Lichman 2013) consists of 57 continuous attributes. These are all statistics computed from e-mail messages, such as word frequencies and average length of uninterrupted sequences of capital letters. The goal is to classify whether an e-mail is spam or legitimate based on these data. The dataset consists of 4601 samples, of which 1813 are spam and 2788 are legitimate. Again, the dataset was standardized and randomly partitioned into 70% training data, 20% test data and 10% validation data. The MLP consisted of three hidden layers with seven, five and three hidden units respectively, all using $\tanh$ activation functions. The network achieved an accuracy of 93.26% on its test set.

Table 4.3 summarizes the relevant statistics for the Iris, Wine and Spambase tests. Figure 4.2 shows boxplots of the bound deviations detected during these tests.

4.3.3 Discussion

First, it is worth pointing out that no bound violations were ever detected in any of the tests, so empirically the theoretical bound appears to be valid. As in Chapter 3, however, we observe that the theoretical bound becomes less tight as the dimensionality of the input increases. This is apparent both in the tests on random MLPs (compare test 4 to the others) and the tests on trained MLPs (Iris, Wine and Spambase have 4, 13 and 57 attributes, respectively). Also, like before, an increase in sampling interval appears to increase the standard deviation, though the effect is less pronounced.

4.4 Improving the robustness of MLPs

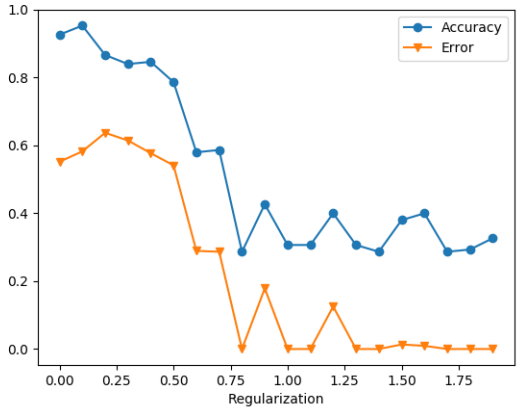
Given the theoretical and empirical evidence for the validity of the bound of Theorem 4.2, it is natural to ask whether it can be used to increase the robustness of MLP classifiers without sacrificing accuracy. The first and most obvious approach that comes to mind is to simply add the theoretical lower bound to the cost function as an additional penalty to be maximized. The drawback, of course, is that the theoretical bound requires computation of the Jacobian of the entire neural network, which quickly becomes infeasible for large networks. A computationally simpler idea is to add $\delta(\boldsymbol{h}_L(\boldsymbol{x}); f)$ to the cost function instead of the full bound. Though less precise, this quantity only depends on the linear classifier applied at the output layer, and maximizing it should clearly increase the lower bound on the robustness as well. The regularized cost function thus becomes

$$\mathcal{L}_1(\boldsymbol{x}, y) = \mathcal{L}(\boldsymbol{x}, y) - \ell\delta(\boldsymbol{h}_L(\boldsymbol{x}); f).$$

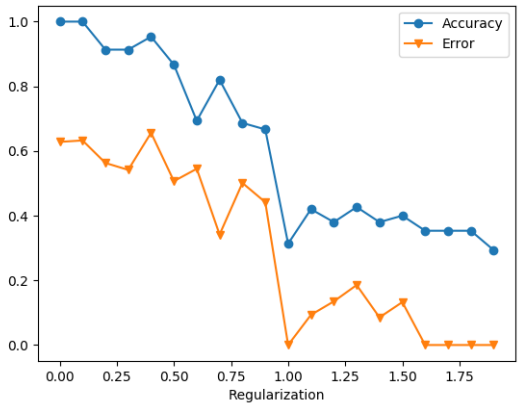
Here, $\ell \in \mathbb{R}$ is the regularization parameter. Figure 4.3 shows the results when tested on the Iris, Wine and Spambase datasets. There are two curves for each plot:

- Accuracy: the accuracy of the network when tested on the entire dataset.
- Error: the error rate of the network. This is the fraction of samples from the dataset that we are able to perturb using FGS or FGV such that it is classified differently.

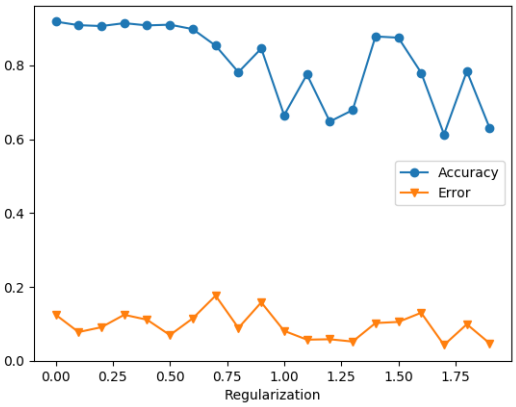
As we can see, the regularization appears to decrease the adversarial error rate, but at a significant cost in accuracy. The Iris and Wine MLPs seem particularly affected by this, but the Spambase MLP can also dip to about 60%, which is unacceptable. Simple regularization thus does not seem like a viable option for increasing robustness...



(a) Iris



(b) Wine



(c) Spambase

Figure 4.3: Accuracy, error rate and minimal adversarial perturbations when regularizing only the output layer.

Chapter 5

Convolutional Neural Networks

Finally, we study Convolutional Neural Networks (CNNs), which are among the most popular state-of-the-art neural network architectures. These networks can be used for all sorts of tasks, but here we will always assume they are used for the classification of images. CNNs can consist of many different types of layers, but crucially they always employ at least one convolution operation (hence their name). The incredible diversity in CNN architectures will force us to adopt a modular approach to analyzing their robustness bounds, where we will study each layer in isolation given only the robustness bounds on the next layer. Although this need not be the case in general, we assume throughout this chapter that CNNs (like MLPs) have a linear classifier at their output layer and thus are at heart linear classifiers of their feature space. This enables us to use the results from the previous chapters to derive lower bounds on the robustness of CNNs.

5.1 Practical examples

Before attempting to analyze the robustness of CNNs to adversarial perturbations, it can be enlightening to gain more intuition about how they work by studying several practical CNNs that have been proposed in the literature.

5.1.1 LeNet-5

LeNet-5 (LeCun et al. 1998a) was one of the first CNNs applied to the task of handwritten digit recognition which outperformed all techniques at the time of its proposal. Figure 5.2 shows a schematic of the network's architecture. The network is constructed to be used on the MNIST dataset, short for Modified National Institute of Standards and Technology (LeCun et al. 1998b). This is a well-known dataset in machine learning. It contains 70,000 grayscale images of size 28×28, each containing a single handwritten digit from 0 to 9. It is considered to be the "hello world" equivalent of machine learning, as it is a relatively simple dataset to work with and train models on. Figure 5.1 shows a few samples from MNIST.

5.1.2 CIFAR-NIN

This network is designed to be used on the CIFAR-10 dataset (Krizhevsky et al. 2009). Named after the Canadian Institute for Advanced Research, this dataset consists of 60,000 32×32 colour images in 10 classes, with 6000 images per class. Figure 5.3 shows a few random samples from this dataset along with their classes. The suffix "NIN" is appended because the network utilizes a technique called Network-in-Network, proposed by Lin et al. 2013. The idea is that instead of only using the traditional convolution-pooling pipeline, small MLPs are used. These are called mlpconv layers, since they are designed to combine

Figure 5.1: Some MNIST samples

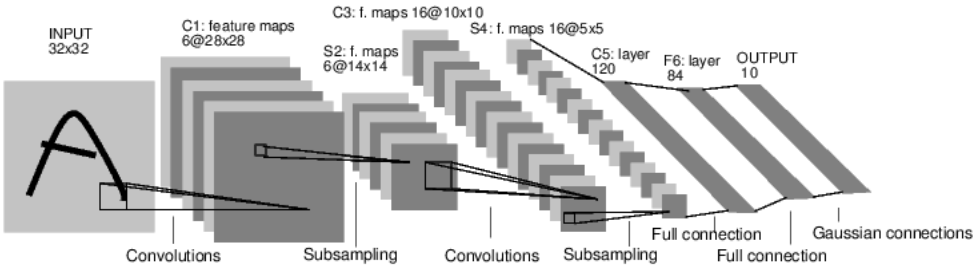


Figure 5.2: Schematic diagram of LeNet-5 architecture.

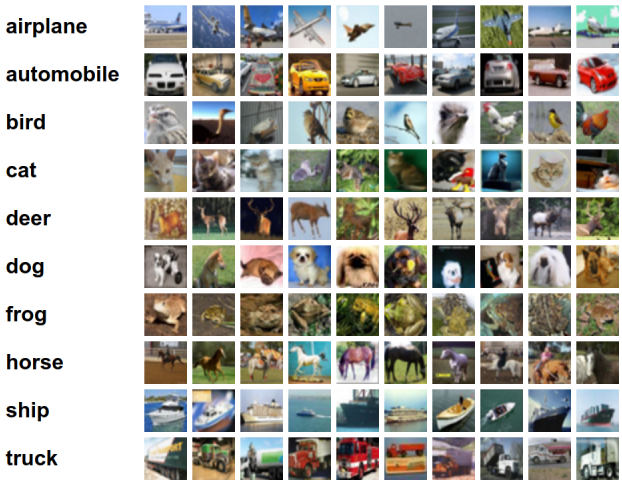


Figure 5.3: Some CIFAR-10 samples

a linear convolution with a Multi-Layer Perceptron. In an ordinary convolution, the feature maps are computed by

$$f_{ijk} = \max\{0, \boldsymbol{w}_k \cdot \boldsymbol{x}_{ij}\}.$$

Here (i, j) is the pixel index in the feature map, $\boldsymbol{x}_{ij}$ stands for the input patch centered at location (i, j), and k is used to index the channels of the feature map. An mlpconv layer, on the other hand, performs the following calculations:

$$\begin{aligned} f^1_{i,j,k_1} &= \max\{0, \boldsymbol{w}_{k_1} \cdot \boldsymbol{x}_{ij} + b_{k_1}\} \\ &\vdots \\ f^n_{i,j,k_n} &= \max\{0, \boldsymbol{w}_{k_n} \cdot \boldsymbol{x}_{ij} + b_{k_n}\} \end{aligned}$$

Here, n denotes the number of layers of the MLP. The network used in Lin et al. 2013 to gauge the performance of their approach on the CIFAR-10 dataset achieved a test error of 10.41% on the test set.

5.1.3 AlexNet

AlexNet (Krizhevsky et al. 2012b) is noteworthy for lighting the spark that set off the deep learning "revolution" by obtaining state-of-the-art performance on the ILSVRC-2012 competition (Russakovsky et al. 2015), where it won first place with an average error rate of 15.3% (the second-best entry had an average error rate of 26.2%). Its architecture is shown schematically in Figure 5.4.

5.1.4 VGGNet

VGGNet (short for Visual Geometry Group Network), introduced by Simonyan et al. 2015, was the first network that managed to push the depth of neural networks to up to 19 layers. This was accomplished by reducing the size of all convolutional kernels to 3×3, whereas previous networks commonly utilized kernels of size 11×11 or 7×7. Simonyan et al. 2015 secured the first and the second places in the localisation and classification tracks respectively of the ImageNet Challenge 2014.

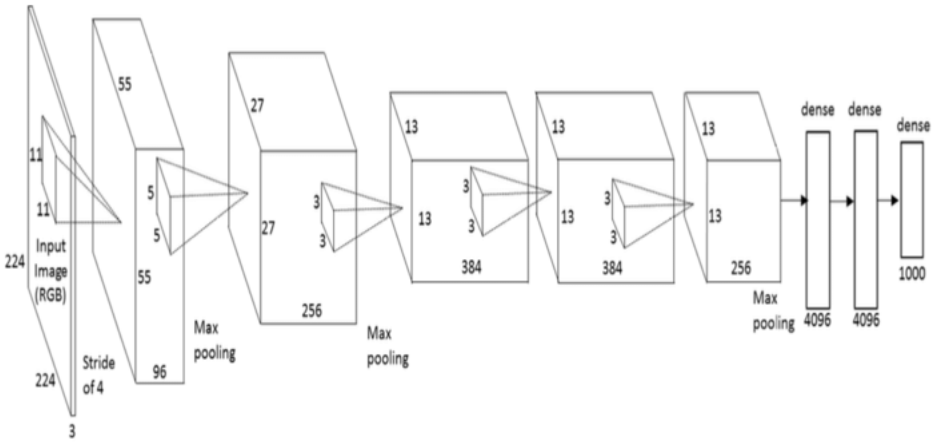


Figure 5.4: AlexNet architecture.

5.1.5 GoogLeNet

GoogLeNet (Szegedy et al. 2015) was responsible for setting the new state of the art for classification and detection in ILSVRC-2014. Figure 5.5 shows a schematic of GoogLeNet architecture. At 27 layers deep, it would not be very instructive to describe its architecture in detail. Instead, we will focus on its most distinguishing feature: the Inception module, depicted in Figure 5.6. An Inception network is a network consisting of Inception modules stacked on top of each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid. One benefit of using Inception modules is computational efficiency: the dimension reducing 1×1 convolutions allow for increasing both the width of each layer as well as the number of layers without an uncontrolled blow-up in computational complexity. Also, by stacking multiple Inception modules on top of each other, the network was able to achieve a great increase in accuracy. Thus, since Inception modules are computationally cheap and stacking them appears to increase accuracy dramatically, GoogLeNet consists of many Inception modules stacked together.

5.1.6 ResNet

ResNet (He et al. 2016), or Deep Residual Networks, is based on the idea that residual mappings are easier to optimize than the original, unreferenced mappings. Formally, if the underlying mapping to be learned by the network is $\boldsymbol{h}(\boldsymbol{x})$, the network is trained to fit the residual $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{x}$. The original mapping is then recast into $\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{x}$. This is implemented in a feedforward neural network using "shortcut connections", depicted in Figure 5.7. These connections simply perform identity mapping, and their output is added to the output of the stacked layers. ResNet is noteworthy for winning the ILSVRC 2015 with a 3.57% top-5 error rate. He et al. 2016 study several residual networks with depths ranging from 34 to 152 layers. Remarkably, the shortcut connections they introduced allowed them to still be able to efficiently train residual networks with up to 152 layers.

5.2 Robustness of CNNs to adversarial perturbations

The biggest problem facing the analysis of robustness of CNNs is the fact that CNN architectures can, in principle, consist of any random combination of different layer types. Comparing the architectures of several practical CNNs such as LeNet (LeCun et al. 1998a), AlexNet (Krizhevsky et al. 2012b), GoogLeNet (Szegedy et al. 2015) and ResNet (He et al. 2016), it would seem the only useful approach is a "modular" one. If we succeed in lower-bounding the robustness of some layer given the robustness of the next layer, we can work our way backwards through the network, starting at the output layer and "backpropagating"

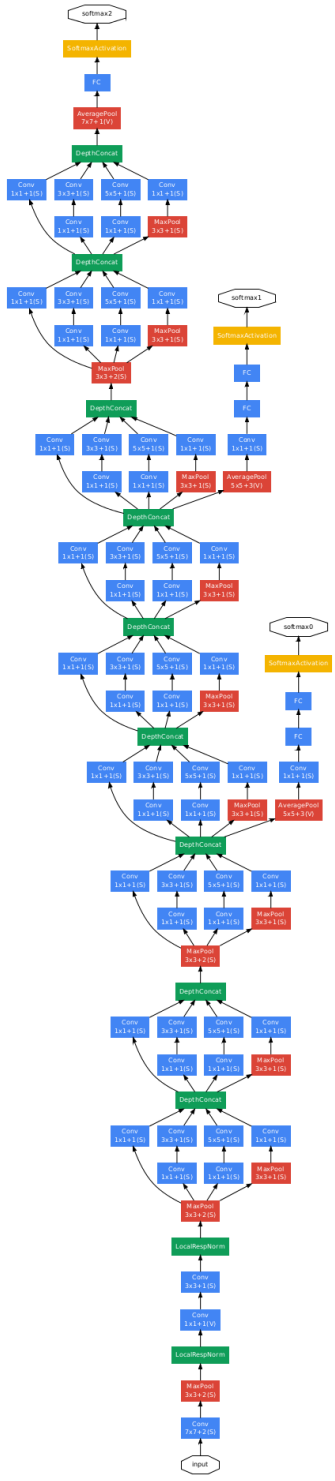


Figure 5.5: GoogLeNet architecture.

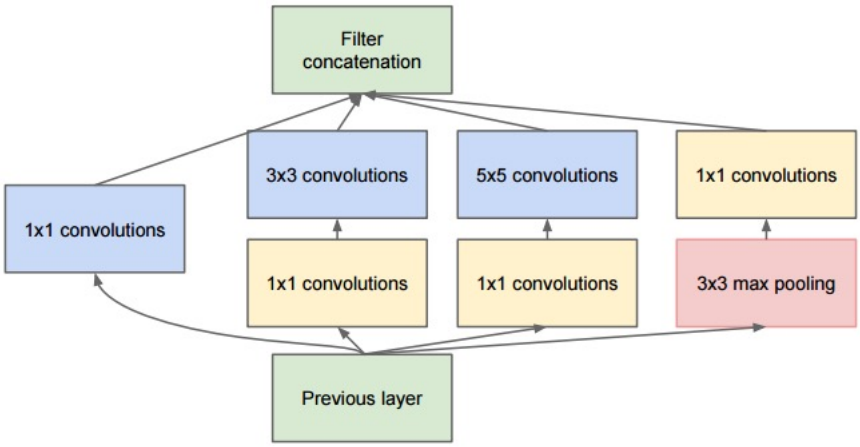


Figure 5.6: Inception module with dimensionality reductions. The reductions are highlighted in yellow; max-pooling is highlighted in red since it is applied only occasionally.

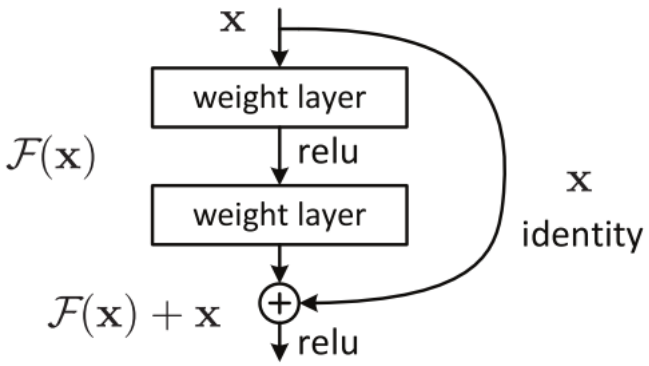


Figure 5.7: Shortcut connections in ResNet.

until we reach the input layer. That way, our approach can be applied to any CNN as long as the robustness bounds of the different layer types have been established. The obvious downside of this idea is that we most likely introduce cumulative approximation errors which increase as the number of layers of the CNN increases. In turn, however, we get a flexible (and hopefully efficient) framework which can handle any CNN architecture composed of known layer types.

5.2.1 Convolutional layers

Suppose we have a convolutional layer whose input consists of a 3D tensor $\mathbf{X} \in \mathbb{R}^{c \times d \times d}$ (the ImageNet dataset, for example, consists of 224×224 RGB images, so a CNN used for ImageNet classification might have convolutional layers where $c = 3$ and $d = 224$). The layer has k kernel tensors $\mathbf{W}_i \in \mathbb{R}^{c \times q \times q}$ ($i = 1, \ldots, k$) with stride s and a bias vector $\boldsymbol{b} \in \mathbb{R}^k$ (i.e. one scalar bias parameter per feature map). In a slight abuse of notation, we will write $\mathbf{W} \star \mathbf{X}$ to denote the output of convolving $\mathbf{X}$ with each of the k kernel tensors $\mathbf{W}_i$. This output is a 3D tensor of size $k \times t \times t$ where $t = \frac{d-q}{s} + 1$ and $(\mathbf{W} \star \mathbf{X})_i = \mathbf{W}_i \star \mathbf{X}$. The different components $(\mathbf{W} \star \mathbf{X})_i$ constitute the feature maps of this convolutional layer. We will also write $\mathbf{W} \star \mathbf{X} + \boldsymbol{b}$ to signify the tensor $\mathbf{W} \star \mathbf{X}$ where each of the k feature maps has its respective bias added in:

$$(\mathbf{W} \star \mathbf{X} + \boldsymbol{b})_{ijk} = (\mathbf{W}_i \star \mathbf{X})_{jk} + b_i.$$

The next layer of the network is assumed to have a robustness bound of κ, in the sense that any adversarial perturbation $\mathbf{Q}$ to $\mathbf{X}$ must satisfy $\|\mathbf{Q}\|_F \geq \kappa$. We can now attempt to bound the norm of a perturbation $\mathbf{R}$ to $\mathbf{X}$ such that

$$\|\mathrm{ReLU}(\mathbf{W} \star (\mathbf{X} + \mathbf{R}) + \boldsymbol{b})\|_F = \|\mathrm{ReLU}(\mathbf{W} \star \mathbf{X} + \boldsymbol{b})\|_F + \kappa. \tag{5.1}$$

Using Lemma A.2 we find[1]

$$\begin{aligned}
\|\mathrm{ReLU}(\mathbf{W} \star (\mathbf{X} + \mathbf{R}) + \boldsymbol{b})\|_F &= \|\mathrm{ReLU}(\mathbf{W} \star \mathbf{X} + \mathbf{W} \star \mathbf{R} + \boldsymbol{b})\|_F \\
&\leq \|\mathrm{ReLU}(\mathbf{W} \star \mathbf{X} + \boldsymbol{b}) + \mathrm{ReLU}(\mathbf{W} \star \mathbf{R})\|_F \\
&\leq \|\mathrm{ReLU}(\mathbf{W} \star \mathbf{X} + \boldsymbol{b})\|_F + \|\mathrm{ReLU}(\mathbf{W} \star \mathbf{R})\|_F .
\end{aligned}$$

This yields

$$\|\mathrm{ReLU}(\mathbf{W} \star \mathbf{R})\|_F \geq \kappa.$$

A necessary condition for this equality to hold is (Lemma A.2)

$$\|\mathbf{W} \star \mathbf{R}\|_F = \kappa. \tag{5.2}$$

Thanks to Lemma A.4, we may rewrite Equation (5.2) as

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{\|\mathbf{W}\|_F}.$$

Hence we finally find

Theorem 5.1. *Consider a convolutional layer with filter tensor $\mathbf{W} \in \mathbb{R}^{k \times c \times q \times q}$ and stride s whose input consists of a 3D tensor $\mathbf{X} \in \mathbb{R}^{c \times d \times d}$. Suppose the next layer has a robustness bound of κ, then any adversarial perturbation to the input of this layer must satisfy*

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{\|\mathbf{W}\|_F}.$$

[1] Note that even though $\mathbf{W} \star (\mathbf{X} + \mathbf{R}) + \boldsymbol{b}$ does not represent a real convolution in this context, the linearity property of Lemma A.2 still applies.

5.2.2 Pooling layers

In the following, assume we have a pooling layer whose pooling operation is p and whose input is $\mathbf{X} \in \mathbb{R}^{c \times d \times d}$. Denote the output of this layer by $\mathbf{Z}(\mathbf{X})$, then we have

$$z_{ijk}(\mathbf{X}) = p(\{x_{i,n+s(j-1),m+s(k-1)} \mid 1 \leq n, m \leq q\}).$$

The output tensor $\mathbf{Z}(\mathbf{X})$ has dimensions $c \times (\frac{d-q}{s} + 1) \times (\frac{d-q}{s} + 1)$. For ease of notation, we let $t = \frac{d-q}{s} + 1$ and we will assume each pooling operation has an associated function I such that

$$z_{ijk}(\mathbf{X}) = p(\{x_{inm} \mid (n, m) \in I(j, k)\}).$$

In the literature, the set $I(j, k)$ is referred to as the *receptive field* of the pooling layer. Each receptive field corresponds to some $q \times q$ region in the input $\mathbf{X}$. To facilitate the analysis of the pooling layers, we make the following assumption:

Assumption 5.2. *The pooling operation satisfies*

$$z_{ijk}(\mathbf{X} + \mathbf{R}) \leq z_{ijk}(\mathbf{X}) + z_{ijk}(\mathbf{R}).$$

Assuming any adversarial perturbation $\mathbf{Q}$ to the input of the next layer needs to satisfy $\|\mathbf{Q}\|_F \geq \kappa$, Assumption 5.2 implies we have to solve the following equation:

$$\|\mathbf{Z}(\mathbf{R})\|_F \geq \kappa. \tag{5.3}$$

A necessary condition for Equation (5.3) to hold is to have

$$|z_{ijk}(\mathbf{R})| \geq \frac{\kappa}{t} \tag{5.4}$$

for at least one element $z_{ijk}(\mathbf{R})$. How this can be done depends on the precise nature of the pooling operation.

MAX-pooling

MAX-pooling reduces the dimensionality of the input by taking the maximum of all $q \times q$ regions within the receptive field:

$$z_{ijk}(\mathbf{X}) = \max\{x_{inm} \mid (n, m) \in I(j, k)\}.$$

Proof that MAX-pooling satisfies Assumption 5.2 is given in Lemma A.5. In order to satisfy Equation (5.4), it is clearly necessary to set at least one component of $\mathbf{R}$ equal to or greater than κ/t in absolute value. This yields

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{t}. \tag{5.5}$$

L_p pooling

An L_p pooling layer produces as output the L_p norm of its input:

$$z_{ijk}(\mathbf{X}) = \left(\sum_{(n,m) \in I} |x_{inm}|^p \right)^{\frac{1}{p}}.$$

Here, we will always assume that $p > 0$. Proof that L_p-pooling satisfies Assumption 5.2 is given in Lemma A.6. In order to satisfy Equation (5.4), we must have

$$\|\boldsymbol{v}_{ijk}(\mathbf{R})\|_p \geq \frac{\kappa}{t} \tag{5.6}$$

for some i, j, k. For Equation (5.6) to be satisfied, there must be at least one element r_{lmn} in some receptive field of $\mathbf{R}$ such that

$$|r_{lmn}| \geq \frac{\kappa}{tq^{2/p}}.$$

Since there will be at least one receptive field, at least one element of $\mathbf{R}$ must satisfy this requirement and hence

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{tq^{2/p}}. \tag{5.7}$$

Note the nice property that as $p \to \infty$ we find

$$\|\boldsymbol{R}\|_F \geq \frac{\kappa}{t},$$

which is the bound for MAX-pooling.

Average pooling

An average pooling layer takes the average of all its inputs:

$$z_{ijk}(\mathbf{X}) = \frac{1}{q^2} \sum_{(n,m)\in I} x_{inm}.$$

Proof that average pooling satisfies Assumption 5.2 is given in Lemma A.7. Note that, contrary to all the other pooling operations studied here, Assumption 5.2 holds with equality in the case of average pooling. To satisfy Equation (5.4), it is necessary that at least one element of $\mathbf{R}$ be greater than or equal to κ/t in absolute value. We thus find

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{t}. \tag{5.8}$$

Finally, we have

Theorem 5.3. *Consider a pooling layer whose operation satisfies Assumption 5.2. Let the input be of size $c \times d \times d$ and the receptive field of size $q \times q$. Let the output be of size $c \times t \times t$. If the robustness bound of the next layer is κ, then the following bounds hold for any adversarial perturbation $\mathbf{R}$:*

- *MAX or average pooling:*

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{t}.$$

- *L_p pooling:*

$$\|\mathbf{R}\|_F \geq \frac{\kappa}{tq^{2/p}}.$$

5.2.3 Batch Normalization layers

Batch Normalization (BN), introduced in Ioffe et al. 2015, aims to eliminate a phenomenon called *internal covariate shift*. The idea is that, during training, the distribution of inputs to the different internal nodes of a neural network is subject to change due to the modification of network parameters, and these changes slow down the convergence of the optimizer. Eliminating this shift can speed up the learning process dramatically. Ioffe et al. 2015 accomplish this by introducing a normalization step which fixes the means and variances of layer inputs. During training, this is done by normalizing and linearly transforming the activations of a mini-batch as follows:

$$y_k = \gamma_k \frac{x_k - \mu_k}{\sqrt{\sigma_k^2 + \varepsilon}} + \beta_k.$$

Here, x_k is the kth component of a sample $\boldsymbol{x} \in \mathbb{R}^d$ from the mini-batch, μ_k and σ_k^2 are the mean and variance, respectively, of the values of the entire mini-batch along the kth dimension, ε is a small non-negative value used to aid numerical stability and γ_k, β_k are learnable parameters. Once the network has been trained, the normalization is changed to

$$y_k = \gamma_k \frac{x_k - \mathrm{E}[x_k]}{\sqrt{\mathrm{Var}[x_k] + \varepsilon}} + \beta_k.$$

In this case, the population statistics are used instead of mini-batch statistics. The resulting vector $\boldsymbol{y}$ is written as

$$\boldsymbol{y} = \mathrm{BN}(\boldsymbol{x}).$$

Assuming the next layer has a robustness of κ, we have to find a perturbation $\boldsymbol{r}$ to $\boldsymbol{x}$ such that

$$\|\mathrm{BN}(\boldsymbol{x} + \boldsymbol{r})\| = \|\mathrm{BN}(\boldsymbol{x})\| + \kappa.$$

Note that

$$\begin{aligned}
\|\mathrm{BN}(\boldsymbol{x} + \boldsymbol{r})\|^2 &= \sum_k \left(y_k^2 + \gamma_k^2 \frac{r_k^2 + 2r_k(x_k - \mathrm{E}[x_k])}{\mathrm{Var}[x_k] + \varepsilon} + 2\gamma_k \beta_k \frac{r_k}{\sqrt{\mathrm{Var}[x_k] + \varepsilon}} \right) \\
&= \|\mathrm{BN}(\boldsymbol{x})\|^2 + \sum_k \left(\gamma_k^2 \frac{r_k^2 + 2r_k(x_k - \mathrm{E}[x_k])}{\mathrm{Var}[x_k] + \varepsilon} + 2\gamma_k \beta_k \frac{r_k}{\sqrt{\mathrm{Var}[x_k] + \varepsilon}} \right).
\end{aligned}$$

Moreover,

$$\|\mathrm{BN}(\boldsymbol{x} + \boldsymbol{r})\|^2 = \|\mathrm{BN}(\boldsymbol{x})\|^2 + 2\kappa \|\mathrm{BN}(\boldsymbol{x})\| + \kappa^2.$$

So we find

$$\begin{aligned}
2\kappa \|\mathrm{BN}(\boldsymbol{x})\| + \kappa^2 &= \sum_k \left(\gamma_k^2 \frac{r_k^2 + 2r_k(x_k - \mathrm{E}[x_k])}{\mathrm{Var}[x_k] + \varepsilon} + 2\gamma_k \beta_k \frac{r_k}{\sqrt{\mathrm{Var}[x_k] + \varepsilon}} \right) \\
&= \sum_k \frac{\gamma_k r_k}{\mathrm{Var}[x_k] + \varepsilon} \left(\gamma_k r_k + 2\gamma_k (x_k - \mathrm{E}[x_k]) + 2\beta_k \sqrt{\mathrm{Var}[x_k] + \varepsilon} \right).
\end{aligned}$$

Let

$$\begin{aligned}
\phi_k &= \frac{\gamma_k}{\mathrm{Var}[x_k] + \varepsilon}, \\
\chi_k &= 2\gamma_k (x_k - \mathrm{E}[x_k]) + 2\beta_k \sqrt{\mathrm{Var}[x_k] + \varepsilon},
\end{aligned}$$

then we may write

$$2\kappa \left\|\mathrm{BN}(\boldsymbol{x})\right\| + \kappa^2 = \sum_k \phi_k r_k(\gamma_k r_k + \chi_k). \tag{5.9}$$

It now follows that in order to satisfy Equation (5.9), there must exist at least one j such that

$$\phi_j r_j(\gamma_j r_j + \chi_j) \geq \frac{2\kappa \left\|\mathrm{BN}(\boldsymbol{x})\right\| + \kappa^2}{d}.$$

Let

$$\psi = \frac{2\kappa \left\|\mathrm{BN}(\boldsymbol{x})\right\| + \kappa^2}{d},$$

then we find the following quadratic inequality in r_j:

$$\phi_j \gamma_j r_j^2 + \phi_j \chi_j r_j - \psi \geq 0. \tag{5.10}$$

The bounds on its solutions are given by

$$\frac{\pm\sqrt{\phi_j^2\chi_j^2 + 4\phi_j\gamma_j\psi} - \phi_j\chi_j}{2\phi_j\gamma_j}.$$

These solutions only exist when $\phi_j^2\chi_j^2 + 4\phi_j\gamma_j\psi \geq 0$, i.e.

$$\gamma_j^2(x_j - \mathrm{E}[x_j])^2 + 2\gamma_j\beta_j(x_j - \mathrm{E}[x_j])\sqrt{\mathrm{Var}[x_j] + \varepsilon} + \beta_j^2(\mathrm{Var}[x_j] + \varepsilon) \geq -\psi(\mathrm{Var}[x_j] + \varepsilon)$$
$$\left(\gamma_j(x_j - \mathrm{E}[x_j]) + \beta_j\sqrt{\mathrm{Var}[x_j] + \varepsilon}\right)^2 \geq -\psi(\mathrm{Var}[x_j] + \varepsilon).$$

This is clearly the case, since the LHS is always positive while the RHS is always negative. Note that it is perfectly legal for r_j to be negative. Since $\phi_j\gamma_j \geq 0$ the solutions are given by

$$r_j \leq \rho_{1,j} = -\frac{\sqrt{\phi_j^2\chi_j^2 + 4\phi_j\gamma_j\psi} + \phi_j\chi_j}{2\phi_j\gamma_j},$$
$$r_j \geq \rho_{2,j} = \frac{\sqrt{\phi_j^2\chi_j^2 + 4\phi_j\gamma_j\psi} - \phi_j\chi_j}{2\phi_j\gamma_j}.$$

Let $\delta_j = \min\{|\rho_{1,j}|, |\rho_{2,j}|\}$.

Lemma 5.4. *For all j, it holds that $|r_j| \geq \delta_j$.*

Proof. Suppose first that $\phi_j\chi_j \geq 0$. Note the following properties:

$$\phi_j\gamma_j \geq 0,$$
$$\sqrt{\phi_j^2\chi_j^2 + 4\phi_j\gamma_j\psi} \geq \phi_j\chi_j,$$
$$\rho_{1,j} \leq 0,$$
$$\rho_{2,j} \geq 0.$$

Hence, in this case, $|\rho_{1,j}| \geq \rho_{2,j}$ and so $\delta_j = \rho_{2,j}$. If $|r_j| < \delta_j$, then $|r_j| < \rho_{2,j}$. This can happen for two reasons:

- $0 < r_j < \rho_{2,j}$.
- $-\rho_{2,j} < r_j < 0$.

In both cases, we find $r_j > \rho_{1,j}$ and $r_j < \rho_{2,j}$, which is a contradiction.

Suppose now that $\phi_j \chi_j < 0$. We have

$$\begin{aligned} \phi_j \gamma_j &\geq 0, \\ \sqrt{\phi_j^2 \chi_j^2 + 4\phi_j \gamma_j \psi} &\geq -\phi_j \chi_j, \\ \rho_{1,j} &\leq 0, \\ \rho_{2,j} &\geq 0. \end{aligned}$$

In this case, $|\rho_{1,j}| \leq \rho_{2,j}$ and so $\delta_j = |\rho_{1,j}|$. If $|r_j| < |\rho_{1,j}|$, this is because of two possible reasons:

- $0 < r_j < |\rho_{1,j}|$.
- $\rho_{1,j} < r_j < 0$.

In both cases, we find $r_j > \rho_{1,j}$ and $r_j < \rho_{2,j}$, which is a contradiction.

We conclude that $|r_j| \geq \delta_j$ for all j. □

Lemma 5.5. *For all j we must have $\|\boldsymbol{r}\| \geq \delta_j$.*

Proof. We know that $|r_j| \leq \|\boldsymbol{r}\|$ and by Lemma 5.4 we have $\delta_j \leq |r_j|$. Hence, $\delta_j \leq |r_j| \leq \|\boldsymbol{r}\|$ for all j. □

A direct corollary of the above results is

Theorem 5.6. *Consider a Batch Normalization layer whose input is given by $\boldsymbol{x} \in \mathbb{R}^d$ and whose output is $\mathrm{BN}(\boldsymbol{x})$. Let the robustness of the next layer be bounded by κ, then any adversarial perturbation $\boldsymbol{r}$ to $\boldsymbol{x}$ must satisfy*

$$\|\boldsymbol{r}\| \geq \max_j \delta_j,$$

where

$$\begin{aligned} \delta_j &= \min\{|\rho_{1,j}|, |\rho_{2,j}|\}, \\ \rho_{1,j} &= -\frac{\sqrt{\phi_j^2 \chi_j^2 + 4\phi_j \gamma_j \psi} + \phi_j \chi_j}{2\phi_j \gamma_j}, \\ \rho_{2,j} &= \frac{\sqrt{\phi_j^2 \chi_j^2 + 4\phi_j \gamma_j \psi} - \phi_j \chi_j}{2\phi_j \gamma_j}, \\ \phi_j &= \frac{\gamma_j}{\mathrm{Var}[x_j] + \varepsilon}, \\ \chi_j &= 2\gamma_j (x_j - \mathrm{E}[x_j]) + 2\beta_j \sqrt{\mathrm{Var}[x_j] + \varepsilon}, \\ \psi &= \frac{2\kappa \, \|\mathrm{BN}(\boldsymbol{x})\| + \kappa^2}{d}, \end{aligned}$$

and γ_j, β_j are the learnable parameters of the Batch Normalization.

It is interesting to explore if this bound may ever be trivial, i.e. equal to zero. For this to happen, there must be a j such that $\rho_{1,j} = 0$ or $\rho_{2,j} = 0$. In both cases, it is necessary that

$$\phi_j^2 \chi_j^2 + 4\phi_j \gamma_j \psi = \phi_j^2 \chi_j^2.$$

This implies $\phi_j \gamma_j \psi = 0$, so at least one of ϕ_j, γ_j or ψ must be zero. For ϕ_j to be zero, we must have $\gamma_j = 0$; for $\psi = 0$, we must have $\kappa = 0$. Assuming (as is realistic) $\kappa > 0$, we find $\gamma_j = 0$. So the bound of Theorem 5.6 is only trivial in case the BN layer itself is trivial along at least one dimension, i.e. when it always outputs a constant β_j regardless of the value of x_j.

5.2.4 Local Response Normalization layers

Local Response Normalization (LRN), introduced by Krizhevsky et al. 2012b, implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. Denoting by a_{ixy} the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity b_{ixy} is given by the expression

$$b_{ixy} = a_{ixy} \left(k + \alpha \sum_{l=\max(0,i-n/2)}^{\min(N-1,i+n/2)} a_{lxy}^2 \right)^{-\beta},$$

where N is the total number of kernels in the layer and the sum runs over n "adjacent" kernel maps at the same spatial position. The constants k, n, α, β are hyperparameters of the LRN. Krizhevsky et al. 2012b used $k = 2, n = 5, \alpha = 10^{-4}$ and $\beta = 0.75$ in their AlexNet CNN.

For an input tensor $\mathbf{X} \in \mathbb{R}^{c \times d \times d}$, the result of applying LRN to $\mathbf{X}$ is written as $\mathrm{LRN}(\mathbf{X})$. It is our objective to find a perturbation $\mathbf{R} \in \mathbb{R}^{c \times d \times d}$ such that

$$\|\mathrm{LRN}(\mathbf{X} + \mathbf{R})\|_F = \|\mathrm{LRN}(\mathbf{X})\|_F + \kappa, \tag{5.11}$$

where κ is the robustness of the next layer. We can make use of Taylor's theorem to find

$$\mathrm{LRN}(\mathbf{X} + \mathbf{R})_{ixy} = \mathrm{LRN}(\mathbf{X})_{ixy} + \nabla_{\boldsymbol{x}} \mathrm{LRN}(\mathbf{X})_{ixy} \cdot \boldsymbol{r} + \varepsilon_{ixy},$$

where we have shortened $\overline{\mathbf{X}}$ to $\boldsymbol{x}$ and $\overline{\mathbf{R}}$ to $\boldsymbol{r}$, and

$$|\varepsilon_{ixy}| \leq \frac{M_{ixy}}{2} \|\boldsymbol{r}\|^2.$$

Here, M_{ixy} is a real number satisfying

$$|\partial^\alpha \mathrm{LRN}(\mathbf{Z})_{ixy}| \leq M_{ixy}$$

for all $\mathbf{Z}$ and $|\alpha| = 2$. Note that $\|\boldsymbol{r}\| = \|\mathbf{R}\|_F$, so we obtain (using the Cauchy-Schwarz inequality)

$$|\mathrm{LRN}(\mathbf{X} + \mathbf{R})_{ixy}| \leq |\mathrm{LRN}(\mathbf{X})_{ixy}| + \|\nabla_{\boldsymbol{x}} \mathrm{LRN}(\mathbf{X})_{ixy}\| \|\mathbf{R}\|_F + \frac{M_{ixy}}{2} \|\mathbf{R}\|_F^2.$$

By Equation (5.11) we know at least one component (i, x, y) of $\mathrm{LRN}(\mathbf{X} + \mathbf{R})$ must satisfy

$$|\mathrm{LRN}(\mathbf{X} + \mathbf{R})_{ixy}| \geq \frac{\|\mathrm{LRN}(\mathbf{X})\|_F + \kappa}{d\sqrt{c}}.$$

We thus find a quadratic inequality in $\|\mathbf{R}\|_F$:

$$\frac{M_{ixy}}{2} \|\mathbf{R}\|_F^2 + \|\nabla_{\boldsymbol{x}} \mathrm{LRN}(\mathbf{X})_{ixy}\| \|\mathbf{R}\|_F + |\mathrm{LRN}(\mathbf{X})_{ixy}| - \frac{\|\mathrm{LRN}(\mathbf{X})\|_F + \kappa}{d\sqrt{c}} \geq 0. \tag{5.12}$$

For notational simplicity, let

$$\delta_{ixy} = |\mathrm{LRN}(\mathbf{X})_{ixy}| - \frac{\|\mathrm{LRN}(\mathbf{X})\|_F + \kappa}{d\sqrt{c}}.$$

Lemma 5.7. *For at least one (i, x, y) it holds that $\delta_{ixy} < 0$.*

Proof. Suppose $\delta_{ixy} \geq 0$ for all (i, x, y), then

$$|\operatorname{LRN}(\mathbf{X})_{ixy}| \geq \frac{\|\operatorname{LRN}(\mathbf{X})\|_F + \kappa}{d\sqrt{c}}.$$

This implies

$$\|\operatorname{LRN}(\mathbf{X})\|_F \geq \|\operatorname{LRN}(\mathbf{X})\|_F + \kappa.$$

This is impossible since $\kappa > 0$. □

Define

$$\Delta = \{(i, x, y) \mid \delta_{ixy} < 0\}.$$

By Lemma 5.7, Δ is non-empty.

Lemma 5.8. *Equation (5.12) must be satisfied by all components $a \in \Delta$.*

Proof. Let $a \in \Delta$ and suppose Equation (5.12) is not satisfied by a, i.e.

$$\frac{M_a}{2}\|\mathbf{R}\|_F^2 + \|\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_a\|\|\mathbf{R}\|_F + |\operatorname{LRN}(\mathbf{X})_a| < \frac{\|\operatorname{LRN}(\mathbf{X})\|_F + \kappa}{d\sqrt{c}}.$$

Note that since $a \in \Delta$ we have $\delta_a < 0$ and so

$$|\operatorname{LRN}(\mathbf{X})_a| < \frac{\|\operatorname{LRN}(\mathbf{X})\|_F + \kappa}{d\sqrt{c}}.$$

This is a necessary condition for Equation (5.12) to be violated by a. We now find

$$|\varepsilon_a| + \|\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_a\|\|\mathbf{R}\|_F + |\operatorname{LRN}(\mathbf{X})_a| < |\operatorname{LRN}(\mathbf{X} + \mathbf{R})_a|.$$

However, this is clearly contradicted by the Taylor expansion of $|\operatorname{LRN}(\mathbf{X} + \mathbf{R})_a|$. □

Theorem 5.9. *Any adversarial perturbation $\mathbf{R}$ to an LRN layer must satisfy*

$$\|\mathbf{R}\|_F \geq \max_{a \in \Delta} \lambda_a,$$

where

$$\lambda_a = \frac{\sqrt{\|\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_a\|^2 - 2M_a\delta_a} - \|\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_a\|}{M_a}.$$

Proof. Let $a \in \Delta$, then by Lemma 5.8 we must have

$$\frac{M_a}{2}\|\mathbf{R}\|_F^2 + \|\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_a\|\|\mathbf{R}\|_F + \delta_a \geq 0$$

The only viable solution to this inequality is given by

$$\|\mathbf{R}\|_F \geq \frac{\sqrt{\|\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_a\|^2 - 2M_a\delta_a} - \|\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_a\|}{M_a}.$$

This inequality holds for arbitrary $a \in \Delta$, from which the result follows. □

The components of $\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_a$ are easy to determine. Let $j \neq a$, then

$$\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_{a,j} = -2x_j x_a \alpha\beta \left(k + \alpha \sum_l x_l^2\right)^{-\beta-1}.$$

Furthermore,

$$\nabla_{\boldsymbol{x}} \operatorname{LRN}(\mathbf{X})_{a,a} = -2x_a^2 \alpha\beta \left(k + \alpha \sum_l x_l^2\right)^{-\beta-1} + \left(k + \alpha \sum_l x_l^2\right)^{-\beta}.$$

The only obstacle to computing the bound from Theorem 5.9 is an efficient algorithm for determining M_a for all $a \in \Delta$. To do this, we first make the following assumption:

Assumption 5.10. *For any LRN layer it holds that $\alpha > 0$, $0 < \beta < 1$ and $k > 0$.*

Assumption 5.10 holds for AlexNet and GoogLeNet (in fact, GoogLeNet uses the exact same hyperparameters for LRN as AlexNet), so it is not entirely unrealistic. If Assumption 5.10 holds, we find

Lemma 5.11. *For any $a \in \Delta$ we have*

$$M_a \leq 4x_{\max}\alpha\beta\left(x_{\max}^2 \alpha(\beta+1)k^{-\beta-2} + k^{-\beta-1}\right) + 2x_{\max}\alpha\beta k^{-\beta-1},$$

where

$$x_{\max} = \max_{i,x,y} |x_{ixy}|.$$

Proof. Let $a \in \Delta$ and let α be a multi-index with $|\alpha| = 2$. Note it is always the case that

$$\max\{0, i - n/2\} \leq i \leq \min\{N-1, i + n/2\}.$$

There are several cases to consider:

- $\alpha_{j_1} = \alpha_{j_2} = 1$ where $j_1 \neq j_2$.
  - $j_1 \neq a$ and $j_2 \neq a$. Then we have

$$\begin{aligned}
\partial^\alpha \operatorname{LRN}(\mathbf{X})_a &= \frac{\partial^2}{\partial x_{j_1} \partial x_{j_2}} \left(x_a \left(k + \alpha \sum_l x_l^2\right)^{-\beta}\right) \\
&= x_a \frac{\partial^2}{\partial x_{j_1} \partial x_{j_2}} \left(k + \alpha \sum_l x_l^2\right)^{-\beta} \\
&= -x_a \alpha\beta \left(k + \alpha \sum_l x_l^2\right)^{-\beta-1} \frac{\partial^2}{\partial x_{j_1} \partial x_{j_2}} \sum_l x_l^2 \\
&= 0.
\end{aligned}$$

  - $j_1 = a$. Then we have

$$\partial^\alpha \operatorname{LRN}(\mathbf{X})_a = \frac{\partial^2}{\partial x_a \partial x_{j_2}} \left(x_a \left(k + \alpha \sum_l x_l^2\right)^{-\beta}\right)$$

First, we compute

$$\frac{\partial}{\partial x_a}\left(x_a\left(k+\alpha\sum_l x_l^2\right)^{-\beta}\right) = x_a\frac{\partial}{\partial x_a}\left(k+\alpha\sum_l x_l^2\right)^{-\beta} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta}$$
$$= -x_a\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\frac{\partial}{\partial x_a}\left(k+\alpha\sum_l x_l^2\right) + \left(k+\alpha\sum_l x_l^2\right)^{-\beta}$$
$$= -2x_a^2\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta}.$$

Then,

$$\frac{\partial}{\partial x_{j_2}}\left(-2x_a^2\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta}\right) =$$
$$-2x_a^2\alpha\beta\frac{\partial}{\partial x_{j_2}}\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1} + \frac{\partial}{\partial x_{j_2}}\left(k+\alpha\sum_l x_l^2\right)^{-\beta} =$$
$$4x_a^2x_{j_2}\alpha^2\beta(\beta+1)\left(k+\alpha\sum_l x_l^2\right)^{-\beta-2} - 2x_{j_2}\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}.$$

Taking absolute values, we get

$$|\partial^\alpha \operatorname{LRN}(\mathbf{X})_a| \le 2x_{\max}\alpha\beta(2x_{\max}^2\alpha(\beta+1)k^{-\beta-2} + k^{-\beta-1}).$$

– $j_2 = a$. Then we have

$$\partial^\alpha \operatorname{LRN}(\mathbf{X})_a = \frac{\partial^2}{\partial x_{j_1}\partial x_a}\left(x_a\left(k+\alpha\sum_l x_l^2\right)^{-\beta}\right)$$

We first compute

$$\frac{\partial}{\partial x_{j_1}}\left(x_a\left(k+\alpha\sum_l x_l^2\right)^{-\beta}\right) = -x_a\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\frac{\partial}{\partial x_{j_1}}\left(k+\alpha\sum_l x_l^2\right)$$
$$= -2x_{j_1}x_a\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}.$$

Then

$$\frac{\partial}{\partial x_a}\left(-2x_{j_1}x_a\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right) =$$
$$-2x_{j_1}\alpha\beta\left(x_a\frac{\partial}{\partial x_a}\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right) =$$
$$-2x_{j_1}\alpha\beta\left(-2x_a^2\alpha(\beta+1)\left(k+\alpha\sum_l x_l^2\right)^{-\beta-2} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right).$$

Taking absolute values:

$$|\partial^\alpha \operatorname{LRN}(\mathbf{X})_a| \le 2x_{\max}\alpha\beta\left(2x_{\max}^2\alpha(\beta+1)k^{-\beta-2} + k^{-\beta-1}\right).$$

- $\alpha_j = 2$.
  - $j \neq a$. Then

$$\partial^\alpha \operatorname{LRN}(\mathbf{X})_a = \frac{\partial^2}{\partial x_j^2}\left(x_a\left(k+\alpha\sum_l x_l^2\right)^{-\beta}\right)$$
$$= x_a \frac{\partial^2}{\partial x_j^2}\left(k+\alpha\sum_l x_l^2\right)^{-\beta}$$

    First, compute

$$\frac{\partial}{\partial x_j}\left(k+\alpha\sum_l x_l^2\right)^{-\beta} = -2x_j\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}$$

    Then

$$\frac{\partial}{\partial x_j}\left(-2x_j\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right) = -2\alpha\beta\frac{\partial}{\partial x_j}\left(x_j\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right)$$
$$= -2\alpha\beta\left(x_j\frac{\partial}{\partial x_j}\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right)$$
$$= -2\alpha\beta\left(-2x_j^2\alpha(\beta+1)\left(k+\alpha\sum_l x_l^2\right)^{-\beta-2} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right).$$

    Thus

$$\partial^\alpha \operatorname{LRN}(\mathbf{X})_a = -2x_a\alpha\beta\left(-2x_j^2\alpha(\beta+1)\left(k+\alpha\sum_l x_l^2\right)^{-\beta-2} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right).$$

    Taking absolute values:

$$|\partial^\alpha \operatorname{LRN}(\mathbf{X})_a| \leq 2x_{\max}\alpha\beta\left(2x_{\max}^2\alpha(\beta+1)k^{-\beta-2} + k^{-\beta-1}\right).$$

  - $j = a$. Then

$$\partial^\alpha \operatorname{LRN}(\mathbf{X})_a = \frac{\partial^2}{\partial x_a^2}\left(x_a\left(k+\alpha\sum_l x_l^2\right)^{-\beta}\right)$$

    First compute

$$\frac{\partial}{\partial x_a}\left(x_a\left(k+\alpha\sum_l x_l^2\right)^{-\beta}\right) = -2x_a^2\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1} + \left(k+\alpha\sum_l x_l^2\right)^{-\beta}.$$

Then

$$\frac{\partial}{\partial x_a}\left(-2x_a^2\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}+\left(k+\alpha\sum_l x_l^2\right)^{-\beta}\right)=$$
$$\frac{\partial}{\partial x_a}\left(-2x_a^2\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right)+\frac{\partial}{\partial x_a}\left(k+\alpha\sum_l x_l^2\right)^{-\beta}=$$
$$-2\alpha\beta\left(x_a^2\frac{\partial}{\partial x_a}\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}+2x_a\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right)-2x_a\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}=$$
$$-2\alpha\beta\left(-2x_a^3\alpha(\beta+1)\left(k+\alpha\sum_l x_l^2\right)^{-\beta-2}+2x_a\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}\right)-2x_a\alpha\beta\left(k+\alpha\sum_l x_l^2\right)^{-\beta-1}.$$

Taking absolute values:

$$|\partial^\alpha \operatorname{LRN}(\mathbf{X})_a| \le 4x_{\max}\alpha\beta\left(x_{\max}^2\alpha(\beta+1)k^{-\beta-2}+k^{-\beta-1}\right)+2x_{\max}\alpha\beta k^{-\beta-1}.$$

Hence we find

$$M_a \le \max\left\{\begin{array}{l} 2x_{\max}\alpha\beta(2x_{\max}^2\alpha(\beta+1)k^{-\beta-2}+k^{-\beta-1}),\\ 4x_{\max}\alpha\beta\left(x_{\max}^2\alpha(\beta+1)k^{-\beta-2}+k^{-\beta-1}\right)+2x_{\max}\alpha\beta k^{-\beta-1}\end{array}\right\}.$$

It is easy to see that this implies

$$M_a \le 4x_{\max}\alpha\beta\left(x_{\max}^2\alpha(\beta+1)k^{-\beta-2}+k^{-\beta-1}\right)+2x_{\max}\alpha\beta k^{-\beta-1}.$$

□

5.2.5 Flattening

The feature maps of the convolutional and pooling layers must be flattened before they can be passed into an MLP. The robustness of MLPs is characterized in terms of the Euclidean norm of the minimal perturbation necessary to change the output classification. Hence, if the robustness of the MLP is bounded by κ and the input is a flattened $c \times d \times d$ tensor $\overline{\mathbf{X}}$, then any adversarial perturbation $\mathbf{Q}$ to $\mathbf{X}$ must satisfy

$$\|\overline{\mathbf{Q}}\| \ge \kappa.$$

By Lemma A.2 this is equivalent to

$$\|\mathbf{Q}\|_F \ge \kappa.$$

5.2.6 Computing the robustness bounds of a CNN

Using the formulas derived in the previous sections, it is possible to construct an algorithm for computing a lower bound on the norm of the smallest perturbation that has to be applied to the input matrix in order to yield a different classification at the output. Algorithm 5.1 shows how this can be done.

5.3 Experimental results

For our experiments, we focus on two networks: LeNet-5 and CIFAR-NIN. Both of these networks are available in the Caffe framework (Jia et al. 2014), of which we made grateful use throughout the following

Algorithm 5.1: Computing robustness bounds of a CNN

Data: A CNN f with L layers and an input $\mathbf{X}$ to f.
Result: A lower bound on the Frobenius norm of any $\mathbf{R}$ such that $f(\mathbf{X}+\mathbf{R}) \neq f(\mathbf{X})$.
begin
  Let κ_L be a lower bound on the adversarial robustness of the MLP at the end of the CNN.
  foreach *layer l of the CNN from $L-1$ to 1* **do**
    Let κ_l be a lower bound on the Frobenius norm of any adversarial perturbation, computed using κ_{l+1} and the inequalities given above.
  end
  return κ_1
end

Network	Normalized theoretical bound	Normalized empirical bound	Rate
LeNet-5	7.274294e−8	0.9334476	99.94%
CIFAR-NIN	1.579417e−17	0.3129927	93.04%

Table 5.1: Theoretical and empirical robustness bounds on different convolutional networks

sections for carrying out tests and running experiments. Caffe (short for Convolutional Architecture for Fast Feature Embedding) provides a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models. The framework is a BSD-licensed C++ library with Python and MATLAB bindings for training and deploying general-purpose convolutional neural networks and other deep models efficiently on commodity architectures.

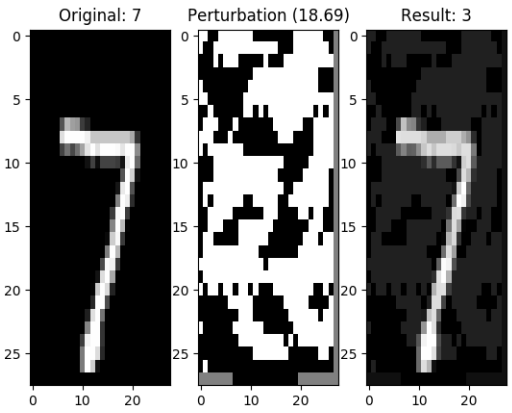
Table 5.1 shows theoretical and empirical bounds on the Frobenius norms of adversarial perturbations for LeNet-5 and CIFAR-NIN. These bounds were computed by averaging over all samples in their respective test sets (MNIST for LeNet-5 and CIFAR-10 for CIFAR-NIN) and then normalized by dividing by the number of pixels in the input (i.e. 28×28 for LeNet and $3 \times 32 \times 32$ for CIFAR-NIN). The table also shows the percentage of samples we were able to adversarially perturb using FGS. Figure 5.11 shows boxplots of ε values used in FGS; Table 5.2 summarizes the statistics.

5.3.1 Discussion

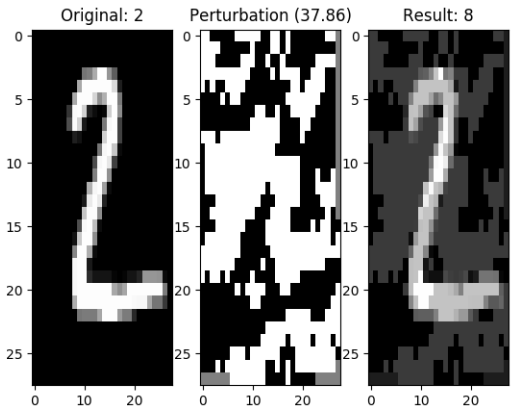
LeNet-5 Due to the pixel values of MNIST samples lying in the interval $[0, 255]$, the epsilon for FGS on LeNet-5 was upper-bounded by 100 instead of 1. Figure 5.8 shows a selection of samples that were adversarially perturbed in the tests for LeNet-5. Figure 5.10a shows a heatmap of the effects of adversarial perturbations on LeNet-5 classification. Of the 10,000 samples in the MNIST test set, only six were not misclassified after perturbation. Curiously, these six samples all belong to the class "8", which coincidently is also the class with the highest number of perturbed samples: 4,767 perturbed samples were misclassified as "8", the highest figure by far. This is followed by 1,921 samples for "3" and 1,078 samples for "2".

CIFAR-NIN As with the MNIST dataset, the pixel values of CIFAR-10 samples lie in the interval $[0, 255]$. As such, a maximum ε of 100 was used. Figure 5.10b shows a heatmap of the effects of adversarial perturbations on CIFAR-NIN classification. Ignoring perturbations that had no effect, we find that the most common mistakes are confusing class 8 ("ship") with class 0 ("airplane"), followed by class 1 ("automobile") with class 9 ("truck") and class 5 ("dog") with class 3 ("bird"). Figure 5.9 shows a selection of adversarial examples for CIFAR-NIN.

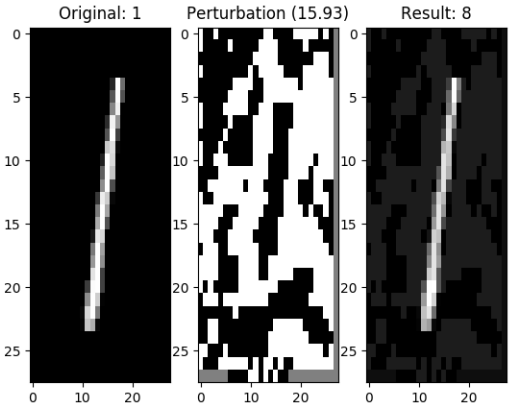
Although the theoretical bounds on average deviate considerably from the perturbations found by FGS, one has to take into consideration that the theoretical bounds were constructed to provide a worst-case



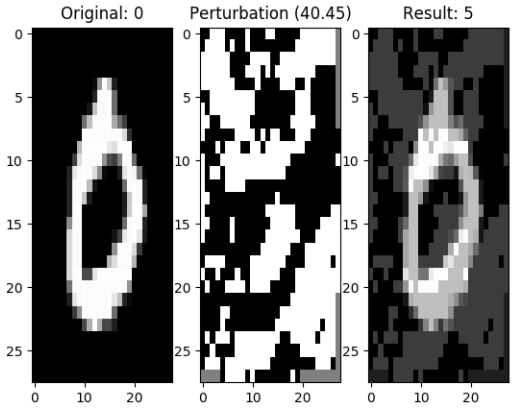
(a) 7 misclassified as 3 with $\varepsilon = 18.69$.



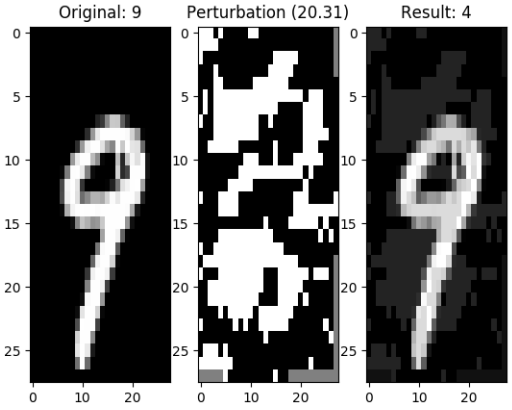
(b) 2 misclassified as 8 with $\varepsilon = 37.86$.



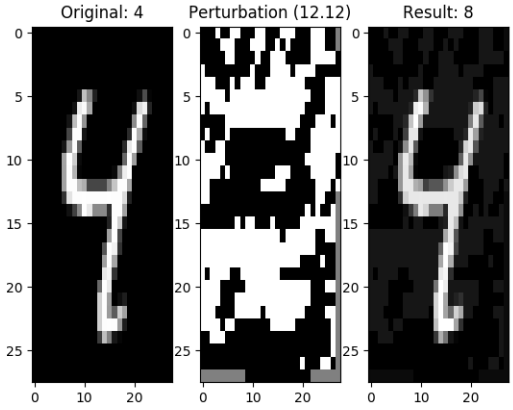
(c) 1 misclassified as 8 with $\varepsilon = 15.93$.



(d) 0 misclassified as 5 with $\varepsilon = 40.45$.

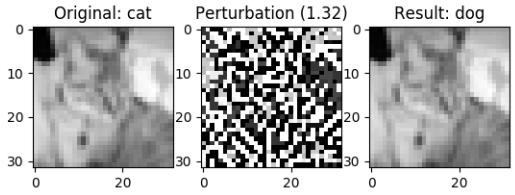


(e) 9 misclassified as 4 with $\varepsilon = 20.31$.

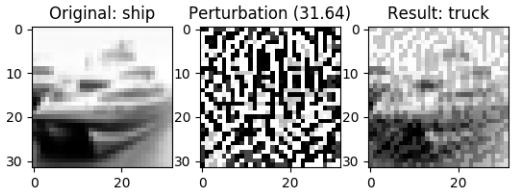


(f) 4 misclassified as 8 with $\varepsilon = 12.12$.

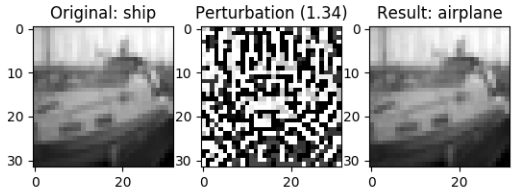
Figure 5.8: Selection of adversarial examples for LeNet-5



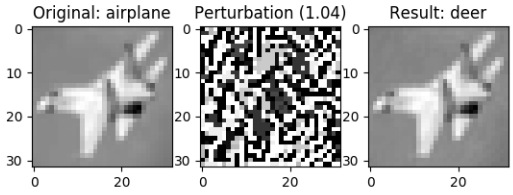
(a) Cat misclassified as dog with $\varepsilon = 1.32$.



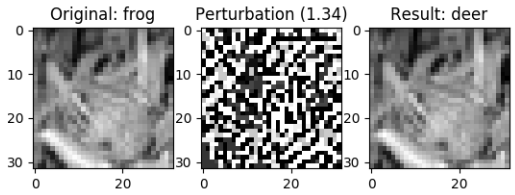
(b) Ship misclassified as truck with $\varepsilon = 31.64$.



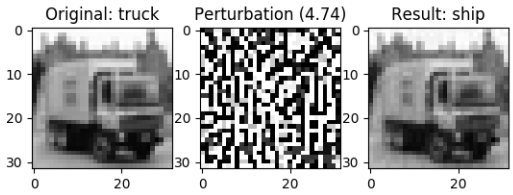
(c) Ship misclassified as airplane with $\varepsilon = 1.34$.



(d) Airplane misclassified as deer with $\varepsilon = 1.04$.

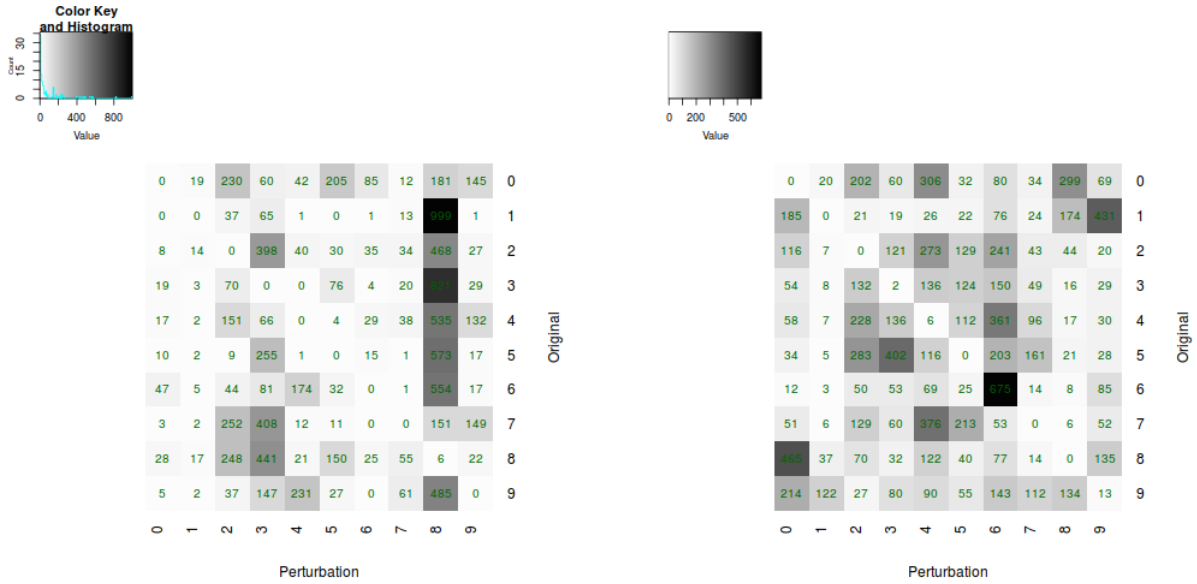


(e) Frog misclassified as deer with $\varepsilon = 1.34$.



(f) Truck misclassified as ship with $\varepsilon = 4.74$.

Figure 5.9: Selection of adversarial examples for CIFAR-NIN

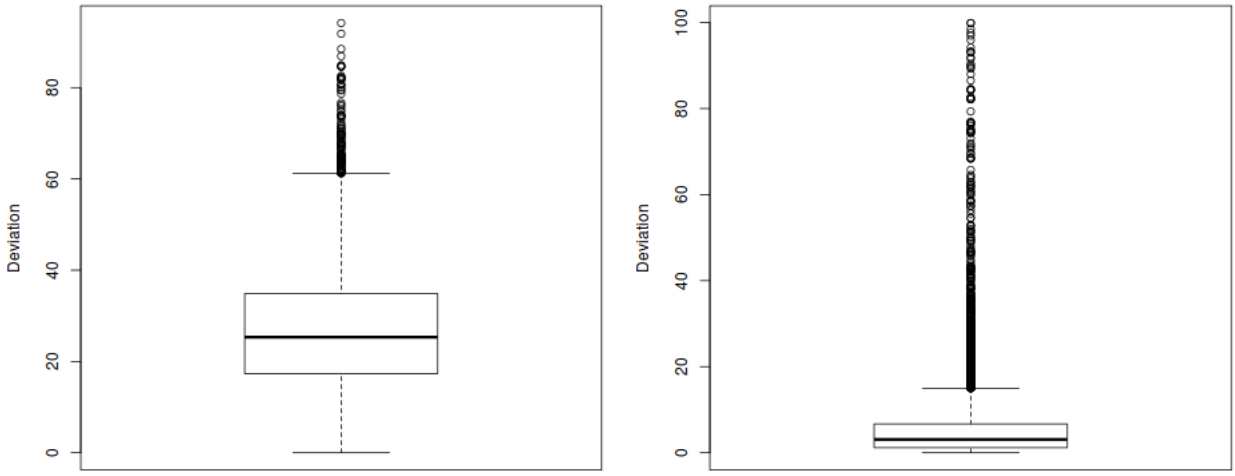


(a) LeNet-5 (b) CIFAR-NIN

Figure 5.10: Heatmap of adversarial perturbations on the different CNNs

Network	Min	Max	Mean	Median	Std
LeNet-5	0.00065	94.17	26.72	25.34	13.27481
CIFAR-NIN	0.00121	99.88000	5.78300	3.09300	9.184525

Table 5.2: Statistics of ε values for the different CNNs



(a) LeNet-5 (b) CIFAR-NIN

Figure 5.11: Boxplots of ε values for the different CNNs

estimate for the norms of adversarial perturbations. These estimates may not hold for all (or even most) input samples. Furthermore, the smallest perturbations we were able to generate on the two data sets have norms that are much closer to the theoretical bound than their averages (0.0182 for MNIST and 0.0671 for CIFAR-10). This indicates that the theoretical bound is not necessarily very loose, but rather that very small adversarial perturbations occur with non-zero probability on natural samples. Note also that the FGS method does not necessarily generate minimal perturbations even with the smallest choice of ε: the method depends on the linearity hypothesis and uses a first-order Taylor approximation of the loss function. Higher-order methods may find much smaller perturbations by exploiting non-linearities in the network, but these are generally much less efficient than FGS.

There is a striking difference in magnitude between MNIST and CIFAR-10 of both the empirical and theoretical perturbations: on average, the perturbations on MNIST are much larger than the ones found for CIFAR-10. This result can be explained by the linearity hypothesis of Goodfellow et al. 2015. The input samples of CIFAR-10 are much larger in dimensionality than MNIST samples, so the linearity hypothesis correctly predicts that networks trained on CIFAR-10 are more susceptible to adversarial perturbations due to the highly linear behavior these classifiers are conjectured to exhibit. However, these differences may also be related to the fact that LeNet-5 achieves much higher accuracy on the MNIST data set than CIFAR-NIN does on CIFAR-10 (over 99% for MNIST compared to about 80% for CIFAR-10).

5.4 Improving the robustness of CNNs

Based on the above results, several architectural recommendations can be formulated which may increase the robustness of neural networks to adversarial perturbations. First, for convolutional layers, it is recommended to keep the dimensionality and absolute values of the components of the kernels as small as possible. This way, the Frobenius norm of the kernels will decrease, and the lower bound on the robustness might increase. For pooling layers, it helps to have a high stride along with small receptive fields. In the case of L_p pooling, p should be as large as possible. Furthermore, the insertion of Batch Normalization and Local Response Normalization layers may also play a role in increasing the robustness. If the CNN has fully-connected layers at the end, it helps to have the components of its Jacobian matrix be small in absolute value. This may be achieved efficiently using, for example, the layer-wise contractive penalty of Gu et al. 2014.

As a concrete illustration of these suggestions, consider the LeNet-5 network. When trained on MNIST, its input has a dimensionality of 28×28. The network then performs the following transformations:

1. C1: a convolution with a kernel size of 5×5, stride of 1 and 20 feature maps;
2. S2: a MAX pooling layer with receptive field size of 2×2 and stride of 2;
3. C3: a convolution with a kernel size of 5×5, stride of 1 and 50 feature maps;
4. S4: a MAX pooling layer with receptive field size of 2×2 and stride of 2;
5. C5: a fully-connected layer with 500 outputs and ReLU activation;
6. F6: a fully-connected layer with 10 outputs;
7. OUTPUT: a softmax layer.

Note that layers C5, F6 and OUTPUT together form an MLP classifier. The robustness of LeNet-5 is thus

given by κ_1, where

$$\kappa_6 = \min_{c' \neq c} \frac{|(\boldsymbol{v}_{c'} - \boldsymbol{v}_c) \cdot \boldsymbol{x} + b_{c'} - b_c|}{\|\boldsymbol{v}_{c'} - \boldsymbol{v}_c\|}, \qquad \kappa_5 = \frac{\sqrt{\|\boldsymbol{J}(\boldsymbol{x})\|^2 + 2M\sqrt{500}\kappa_6} - \|\boldsymbol{J}(\boldsymbol{x})\|}{M\sqrt{500}},$$
$$\kappa_4 = \frac{\kappa_5}{t_2}, \qquad \kappa_3 = \frac{\kappa_4}{\|\mathbf{W}_2\|_F},$$
$$\kappa_2 = \frac{\kappa_3}{t_1}, \qquad \kappa_1 = \frac{\kappa_2}{\|\mathbf{W}_1\|_F}.$$

Suppose now that the components of $\mathbf{W}_1$ and $\mathbf{W}_2$ are scaled by a factor of α_1 and α_2 respectively, where $0 < \alpha_1, \alpha_2 < 1$. Then we have

$$\kappa_3' = \frac{\kappa_4}{\alpha_2 \|\mathbf{W}_2\|_F}, \qquad \kappa_2' = \frac{\kappa_3'}{t_1}, \qquad \kappa_1' = \frac{\kappa_2'}{\alpha_1 \|\mathbf{W}_1\|_F}.$$

Assuming the other layers remain unaffected by this change, we may write

$$\kappa_1 = \frac{\kappa_4}{t_1 \|\mathbf{W}_1\|_F \|\mathbf{W}_2\|_F},$$
$$\kappa_1' = \frac{\kappa_4}{t_1 \alpha_1 \|\mathbf{W}_1\|_F \alpha_2 \|\mathbf{W}_2\|_F}.$$

Here, κ_1' gives a robustness bound on the modified network. The ratio of κ_1' to κ_1 is

$$\frac{\kappa_1'}{\kappa_1} = \frac{1}{\alpha_1 \alpha_2}.$$

Hence, we find that the robustness of the network can be increased arbitrarily by scaling the components of the kernel tensors: as the scaling factors tend to zero, the robustness tends to infinity. This is not surprising: if the scaling factors are set to zero, then the convolutional layers output constant zero tensors. This causes the input to no longer have any effect whatsoever on the output and the network to have an infinite robustness. Of course, one cannot arbitrarily scale the components of the convolutional kernels without affecting accuracy, so while this analysis does show that robustness increases as the components of the kernels become smaller, it does not tell us anything about the effect on accuracy. Most likely, this effect will be negative, but if the network has a high number of layers, we may still be able to achieve significant increases in robustness without sacrificing too much accuracy. Suppose we have a network with n convolutional layers whose kernels are $\mathbf{W}_1, \ldots, \mathbf{W}_n$. If we scale each kernel $\mathbf{W}_i$ with a factor $0 < \alpha_i < 1$ while leaving all other layers alone, then by the above analysis we expect a relative increase in robustness of

$$\frac{1}{\alpha_1 \ldots \alpha_n}.$$

It is reasonable to assume that the factors α_i must be close to 1 if we are to preserve accuracy. Assume for simplicity that we can collectively scale the different kernels by at most $1 - \varepsilon$ for some small ε, then we can achieve an increase of at most $(1 - \varepsilon)^{-n}$ while maintaining an acceptable accuracy. Figure 5.12 plots this curve as a function of n assuming $\varepsilon = 0.1$. We can see that the addition of extra convolutional kernels makes it easier to increase the robustness using this technique: scaling four kernels gives a relative increase of about 1.5; for seven kernels, we get a factor of 2.

These effects bear some similarity to the linearity hypothesis of Goodfellow et al. 2015. According to this hypothesis, CNNs learn highly linear mappings and so adversarial examples exist because small perturbations in the input "snowball" into large perturbations of the output. On the other hand, the above results suggest that small increases in the robustness of many different layers can snowball into a large increase in the robustness of the network as a whole. As modern neural networks tend to become ever deeper, such a result inspires a degree of optimism regarding the problem of adversarial perturbations: the

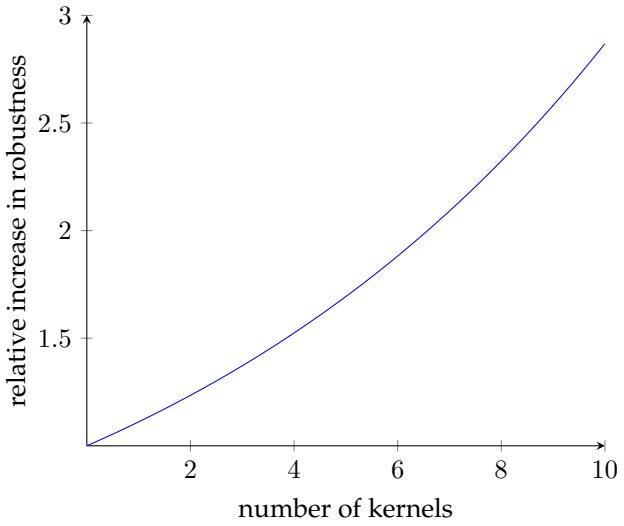


Figure 5.12: The relative increase in theoretical robustness of a CNN as a function of the number of convolutional kernels, where one scales each of the kernel tensors by a factor of 0.9.

more layers a network has, the greater the effect will be of small increases in robustness in the different layers. As an example, consider again the LeNet-5 network and suppose we succeed in increasing the robustness of layer i by a quantity δ. For simplicity, we assume this increase is the same in each layer. Restricting our attention to the convolutional and pooling layers, we find

$$\kappa_4' = \frac{\kappa_5}{t_2} + \delta, \qquad \kappa_3' = \frac{\kappa_4'}{\|\mathbf{W}_2\|_F} + \delta,$$
$$\kappa_2' = \frac{\kappa_3'}{t_1} + \delta, \qquad \kappa_1' = \frac{\kappa_2'}{\|\mathbf{W}_1\|_F} + \delta.$$

This yields

$$\begin{aligned}
\kappa_1' &= \frac{1}{\|\mathbf{W}_1\|_F}\left(\frac{1}{t_1}\left(\frac{1}{\|\mathbf{W}_2\|_F}\left(\frac{\kappa_5}{t_2}+\delta\right)+\delta\right)+\delta\right)+\delta \\
&= \frac{\kappa_5}{t_1 t_2 \|\mathbf{W}_1\|_F \|\mathbf{W}_2\|_F} + \delta\left(\frac{1}{t_1 \|\mathbf{W}_1\|_F \|\mathbf{W}_2\|_F} + \frac{1}{t_1 \|\mathbf{W}_1\|_F} + \frac{1}{\|\mathbf{W}_1\|_F} + 1\right) \\
&= \kappa_1 + \delta\left(\frac{\kappa_1}{\kappa_4} + \frac{\kappa_1}{\kappa_3} + \frac{\kappa_1}{\kappa_2} + 1\right) \\
&= \kappa_1\left(1 + \delta\left(\frac{1}{\kappa_4} + \frac{1}{\kappa_3} + \frac{1}{\kappa_2} + \frac{1}{\kappa_1}\right)\right).
\end{aligned}$$

The relative increase in robustness is thus given by

$$\frac{\kappa_1'}{\kappa_1} = 1 + \delta\left(\frac{1}{\kappa_4} + \frac{1}{\kappa_3} + \frac{1}{\kappa_2} + \frac{1}{\kappa_1}\right).$$

More generally, if we consider a network consisting solely of convolutional and pooling layers, we find

$$\frac{\kappa_1'}{\kappa_1} = 1 + \delta\sum_{l=1}^{L}\frac{1}{\kappa_l}. \tag{5.13}$$

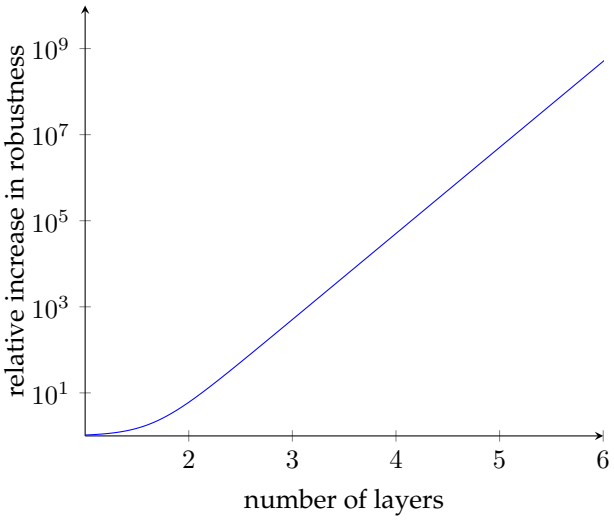


Figure 5.13: The relative increase in theoretical robustness of a CNN as a function of the number of layers, where one increases the robustness of each individual layer by $\delta = 0.1$ and the output layer has a robustness of 2. It is assumed that $\kappa_l = 0.01\kappa_{l+1}$ for all $l < L$.

If we assume $\kappa_i = \alpha\kappa_{i+1}$ for all $i < L$ and $\alpha \neq 1$, then $\kappa_i = \alpha^{L-i}\kappa_L$ and

$$\frac{\kappa_1'}{\kappa_1} = 1 + \frac{\delta}{\kappa_L}\sum_{l=1}^{L}\frac{1}{\alpha^{L-l}} = 1 + \frac{\delta}{\kappa_L}\sum_{l=0}^{L-1}\frac{1}{\alpha^{l}} = 1 + \frac{\delta(1-\alpha^{-L})}{\kappa_L(1-\alpha^{-1})}.$$

This relationship is linear in δ but exponential in L. Figure 5.13 shows a plot of the relative increase in theoretical robustness as a function of the number of layers for $\alpha = 0.01$, $\delta = 0.1$ and $\kappa_L = 2$. Note that for this exponential relationship to hold, it is necessary that $0 < \alpha < 1$, i.e. that the robustness of layer l is smaller than the robustness of layer $l + 1$. Otherwise, the relative increase in robustness saturates at a certain point, meaning that the addition of extra layers will have a negligible effect on the robustness of the network. The relative increase at this saturation point is given by

$$\lim_{L\to\infty} 1 + \frac{\delta(1-\alpha^{-L})}{\kappa_L(1-\alpha^{-1})} = 1 + \frac{\delta}{\kappa_L(1-\alpha^{-1})}.$$

We conclude that it is possible to significantly increase the robustness of a neural network by only slightly increasing the robustness of its individual layers. Moreover, this effect grows exponentially in the number of layers. Thus, although the linearity hypothesis indicates that deeper networks are more vulnerable to adversarial perturbations, our analysis suggests that it is actually easier to increase the robustness of a deep network than the robustness of a shallow one.

5.4.1 Experimental results

In this section, we explore the effect of adding Local Response Normalization layers on the robustness of LeNet-5. Specifically, we create three variants of the original LeNet-5:

- LeNet-LRN1. This is LeNet-5 with a single LRN layer after the first pooling operation.
- LeNet-LRN2. This is LeNet-5 with a single LRN layer after the second pooling operation.
- LeNet-LRN3. This is LeNet-5 with two LRN layers, one after each pooling operation.

Network	Accuracy	Min	Max	Mean	Median	Std	Rate
LeNet-LRN1	98.76%	0.00756	99.71930	25.31045	21.88714	17.30748	98.36%
LeNet-LRN2	98.48%	0.00011	99.87615	24.92861	19.62794	20.60854	95.52%
LeNet-LRN3	98.94%	0.00869	99.95050	25.76302	20.96658	18.83517	96.94%

Table 5.3: Summary of ε values for the LeNet-LRN networks

All of these networks were trained for $10,000$ iterations using the Adam optimization algorithm (Kingma et al. 2014). The results are summarized in Table 5.3. From these results, we can conclude that the introduction of LRN layers into the network can indeed lower the adversarial error rate as well as increase the robustness of the network. However, the precise location of the LRN layers plays an important role. Specifically, the results seem to indicate that LRN layers placed farther towards the end of the network have a diminished effect on robustness and may even cause the robustness to decrease. LeNet-LRN1 and LeNet-LRN3 have a significantly larger minimal perturbation, though, and all networks have a lower adversarial error rate than the original LeNet-5. The accuracy also does not seem to have suffered much from these modifications, suggesting that normalization layers (LRN, BN or otherwise) are a viable technique for increasing robustness without sacrificing accuracy.

Chapter 6

Conclusion

In this thesis, we have attempted to characterize the precise nature and cause of adversarial perturbations. This was done by deriving lower bounds on the magnitude of the minimal perturbations necessary to change the classification of a given classifier. Whereas other related works such as Fawzi et al. 2016 do this by considering general classifiers $f : \mathbb{R}^d \to \{1, \ldots, C\}$, here we have focused our attention on specific families of classifiers and attempted to relate their robustness directly to their model parameters and hyperparameters. We studied linear classifiers, Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). Due to the broad range of architectures used in modern CNNs, the analysis for these networks was performed in a modular manner: we considered each layer in isolation and derived lower bounds on the robustness of a layer given only the robustness of the next layer. We empirically verified the theoretical bounds on every family of classifiers and found no violations. The results of our analysis seem to suggest that it is possible to significantly increase the robustness of a neural network by slight increments in the robustness of its individual layers, and this effect grows exponentially with the number of layers.

The proposed bounds can be computed efficiently, requiring an amount of operations that grows linearly with the number of model parameters and hyperparameters. This makes them useful for several applications, such as

- Model selection. When one is confronted with several different models performing the same classification task, choosing the "right" model is non-trivial. With the proposed theoretical robustness bounds, one can efficiently characterize and compare the robustness of the different models to adversarial perturbations without having to solve expensive optimization problems.

- Techniques for improving robustness. The theoretical analysis presented in this work may lead to new ways of increasing classifier robustness to adversarial perturbations without sacrificing too much accuracy. We already suspect that a form of normalization performed at regular intervals within the model may increase robustness, and we verified this claim in the particular case of Local Response Normalization.

6.1 Future work

The theoretical analysis in this work was limited to linear classifiers, Multi-Layer Perceptrons with softmax output layers and Convolutional Neural Networks consisting of convolutional layers, pooling layers, Batch Normalization layers and Local Response Normalization layers. The most obvious avenue for future work is to extend this analysis to other types of layers as well as other architectures, such as the Long Short-Term Memory (Hochreiter et al. 1997).

Another avenue is to fine-tune the bounds so that they deviate less from the actual perturbations found by methods such as FGS, since the bounds proposed here do not appear to be very tight. However, due to the FGS method being only an approximation based on the linearity hypothesis, it is unclear whether

the theoretical bounds deviate from actual perturbations because they are loose or because the FGS method does not find minimal perturbations. Exploring this question further would be interesting, e.g. by comparing the theoretical bounds to perturbations found by more general methods such as L-BFGS.

There are also no error estimates given for the proposed theoretical bounds. Quantifying how much the bounds can deviate from the actual perturbations could improve the estimates and make the bounds tighter. Providing estimates for the expected deviation and the variance of the deviation of the theoretical bounds with respect to the actual bounds would certainly be an improvement.

It is reasonable to expect that there exists a trade-off between classifier accuracy and robustness. For example, if a classifier for MNIST always classifies everything as 0, then that classifier will have an accuracy of about 10% yet its robustness will be infinite. Formally characterizing the relationship between accuracy and robustness is a question we definitely want to explore further. We expect that the robustness of a classifier can be increased without loss of accuracy (or even with an increase in accuracy) up to a point where the robustness equals that of the oracle $f^\star$; increasing the robustness beyond this point should cause the accuracy to decrease as the classifier becomes too insensitive to changes.

Finally, we think it would also be worthwhile to study the trade-off between neural network depth and breadth. It is strongly suspected that deeper networks can be exponentially more efficient than shallow ones (Bengio et al. 2007), but the precise effect of increasing depth vs. increasing breadth on the robustness has not been studied yet to our knowledge. Based on the hypotheses surrounding the cause of adversarial perturbations, however, it is reasonable to expect that shallow networks will in fact be more robust than deeper ones unless specific countermeasures are taken.

Appendix A

Auxiliary results

A.1 Multi-Layer Perceptrons

Theorem A.1 (Taylor's theorem for multivariate vector-valued functions (Spivak 1994)). *Let $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ be a function defined by*

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})]^T,$$

where each f_i is twice differentiable on $\mathbb{R}^n$. Suppose also that there exists an $M \geq 0$ such that each f_i satisfies $|\partial^\alpha f_i(x)| \leq M$ for all $\boldsymbol{x}$ and $|\alpha| = 2$. Then

$$\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{r}) = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})\boldsymbol{r} + \boldsymbol{\varepsilon},$$

where

$$\|\boldsymbol{\varepsilon}\| \leq \frac{M}{2}\sqrt{m}\,\|\boldsymbol{r}\|^2.$$

A.2 Convolutional Neural Networks

Lemma A.2.

1. *Let $a, b \in \mathbb{R}$, then*

$$\mathrm{ReLU}(a + b) \leq \mathrm{ReLU}(a) + \mathrm{ReLU}(b).$$

2. *Let $\mathbf{W}, \mathbf{X}, \mathbf{R}$ be tensors and $s \in \mathbb{N}$, then*

$$\mathbf{W} \star_s (\mathbf{X} + \mathbf{R}) = \mathbf{W} \star_s \mathbf{X} + \mathbf{W} \star_s \mathbf{R}.$$

3. *Let $\mathbf{X}$ be any real-valued tensor, then*

$$\left\|\overline{\mathbf{X}}\right\| = \|\mathbf{X}\|_F.$$

4. *Let $\mathbf{X}$ be any real-valued tensor, then*

$$\|\mathrm{ReLU}(\mathbf{X})\|_F \leq \|\mathbf{X}\|_F.$$

Proof.

1. We distinguish four cases:
   - $a, b > 0$:
   $$\mathrm{ReLU}(a+b) = a+b = \mathrm{ReLU}(a) + \mathrm{ReLU}(b).$$
   - $a > 0$ and $b \leq 0$:
   $$\mathrm{ReLU}(a+b) \leq a+b \leq a = \mathrm{ReLU}(a) + \mathrm{ReLU}(b).$$
   - $a \leq 0$ and $b > 0$:
   $$\mathrm{ReLU}(a+b) \leq a+b \leq b = \mathrm{ReLU}(a) + \mathrm{ReLU}(b).$$
   - $a, b < 0$:
   $$\mathrm{ReLU}(a+b) = 0 = \mathrm{ReLU}(a) + \mathrm{ReLU}(b).$$
2. Suppose $\mathbf{W} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and $\mathbf{X}, \mathbf{R} \in \mathbb{R}^{m_1 \times \ldots m_d}$, then
   $$\begin{aligned}(\mathbf{W} \star_s (\mathbf{X} + \mathbf{R}))_{i_1 \ldots i_d} &= \sum_{j_1,\ldots,j_d} w_{j_1 \ldots j_d}(x_{i_1+s(j_1-1),\ldots,i_d+s(j_d-1)} + r_{i_1+s(j_1-1),\ldots,i_d+s(j_d-1)}) \\ &= \sum_{j_1,\ldots,j_d} w_{j_1 \ldots j_d} x_{i_1+s(j_1-1),\ldots,i_d+s(j_d-1)} + \sum_{j_1,\ldots,j_d} w_{j_1 \ldots j_d} r_{i_1+s(j_1-1),\ldots,i_d+s(j_d-1)} \\ &= (\mathbf{W} \star_s \mathbf{X})_{i_1 \ldots i_d} + (\mathbf{W} \star_s \mathbf{R})_{i_1 \ldots i_d}.\end{aligned}$$
3. This is obvious.
4. Let $\mathbf{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$. We compute:
   $$\|\mathrm{ReLU}(\mathbf{X})\|_F^2 = \sum_{i_1,\ldots,i_d} \max\{0, x_{i_1 \ldots i_d}\}^2 \leq \sum_{i_1,\ldots,i_d} x_{i_1 \ldots i_d}^2 = \|\mathbf{X}\|_F^2 .$$

□

Lemma A.3. *Let a_i, b_i be non-negative real numbers for $i = 1, \ldots, n$. Then*

$$\sum_i a_i b_i \leq \left(\sum_i a_i\right)\left(\sum_i b_i\right).$$

Proof. This is easy to see by direct computation:

$$\left(\sum_i a_i\right)\left(\sum_i b_i\right) = \sum_i a_i \left(\sum_j b_j\right).$$

Since all terms are non-negative, we have

$$b_i \leq \sum_j b_j$$

for all i. Hence the result follows. □

Lemma A.4. *Let* $\mathbf{W}$ *and* $\mathbf{R}$ *be as above, then*

$$\|\mathbf{W} \star \mathbf{R}\|_F \leq \|\mathbf{W}\|_F \|\mathbf{R}\|_F .$$

Proof. We compute:

$$\begin{aligned}\|\mathbf{W} \star \mathbf{R}\|_F^2 &= \sum_i \|\mathbf{W}_i \star \mathbf{R}\|_F^2 = \sum_{i,j,k} (\mathbf{W}_i \star \mathbf{R})_{jk}^2 \\ &= \sum_{i,j,k,l,m,n} w_{ilmn}^2 r_{l,m+s(j-1),n+s(k-1)}^2 .\end{aligned}$$

Using Lemma A.3:

$$\begin{aligned}\sum w_{ilmn}^2 r_{l,m+s(j-1),n+s(k-1)}^2 &\leq \left(\sum w_{ilmn}^2\right)\left(\sum r_{l,m+s(j-1),n+s(k-1)}^2\right) \\ &\leq \left(\sum w_{ilmn}^2\right)\left(\sum r_{lmn}^2\right) \\ &= \|\mathbf{W}\|_F^2 \|\mathbf{R}\|_F^2 .\end{aligned}$$

We can thus conclude

$$\|\mathbf{W} \star \mathbf{R}\|_F \leq \|\mathbf{W}\|_F \|\mathbf{R}\|_F .$$

□

Lemma A.5. *MAX-pooling satisfies Assumption 5.2.*

Proof. We compute:

$$\begin{aligned}z_{ijk}(\mathbf{X} + \mathbf{R}) &= \max\{x_{inm} + r_{inm} \mid (n,m) \in I(j,k)\} \\ &\leq \max\{x_{inm} \mid (n,m) \in I(j,k)\} + \max\{r_{inm} \mid (n,m) \in I(j,k)\} \\ &= z_{ijk}(\mathbf{X}) + z_{ijk}(\mathbf{R}).\end{aligned}$$

□

Lemma A.6. *L_p pooling satisfies Assumption 5.2.*

Proof. Define $\boldsymbol{v}_{ijk}(\mathbf{X})$ to be the vector whose L_p norm is taken in the computation of $z_{ijk}(\mathbf{X})$. We find

$$\begin{aligned}z_{ijk}(\mathbf{X} + \mathbf{R}) &= \|\boldsymbol{v}_{ijk}(\mathbf{X} + \mathbf{R})\|_p = \|\boldsymbol{v}_{ijk}(\mathbf{X}) + \boldsymbol{v}_{ijk}(\mathbf{R})\|_p \\ &\leq \|\boldsymbol{v}_{ijk}(\mathbf{X})\|_p + \|\boldsymbol{v}_{ijk}(\mathbf{R})\|_p = z_{ijk}(\mathbf{X}) + z_{ijk}(\mathbf{R}).\end{aligned}$$

□

Lemma A.7. *Average pooling satisfies Assumption 5.2.*

Proof. We have

$$\begin{aligned}z_{ijk}(\mathbf{X} + \mathbf{R}) &= \frac{1}{q^2} \sum_{(n,m)\in I} (x_{inm} + r_{inm}) \\ &= \frac{1}{q^2} \sum_{(n,m)\in I} x_{inm} + \frac{1}{q^2} \sum_{(n,m)\in I} r_{inm} \\ &= z_{ijk}(\mathbf{X}) + z_{ijk}(\mathbf{R}).\end{aligned}$$

□

Bibliography

Bengio, Yoshua (2009). *Learning Deep Architectures for AI*. Tech. rep. 1312. Université de Montréal.

Bengio, Yoshua and Pascal Lamblin (2007). 'Greedy layer-wise training of deep networks'. In: *Advances in neural ...* 1, pp. 153–160. ISSN: 01628828. DOI: citeulike-article-id:4640046. arXiv: 0500581 [submit]. URL: https://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf.

Byrd, Richard H et al. (1995). 'A limited memory algorithm for bound constrained optimization'. In: *SIAM Journal on Scientific Computing* 16.5, pp. 1190–1208.

Carlini, Nicholas and David Wagner (2016). 'Defensive distillation is not robust to adversarial examples'. In: *arXiv preprint*.

Collins, Francis S, Michael Morgan and Aristides Patrinos (2003). 'The Human Genome Project: lessons from large-scale biology'. In: *Science* 300.5617, pp. 286–290.

Connect, a. K., a. Krogh and J. a. Hertz (1992). 'A Simple Weight Decay Can Improve Generalization'. In: *Advances in Neural Information Processing Systems* 4, pp. 950–957. URL: http://0-citeseerx.ist.psu.edu.innopac.up.ac.za/viewdoc/summary?doi=10.1.1.41.2305.

Dahl, George E., Tara N. Sainath and Geoffrey E. Hinton (2013). 'Improving deep neural networks for LVCSR using rectified linear units and dropout'. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 8609–8613. ISSN: 15206149. DOI: 10.1109/ICASSP.2013.6639346. arXiv: arXiv:1301.3605v3.

Dauphin, Yann N. and Yoshua Bengio (2013). 'Big Neural Networks Waste Capacity'. In: 2011, pp. 1–5. arXiv: 1301.3583. URL: http://arxiv.org/abs/1301.3583.

Deng, Jia et al. (2009). 'Imagenet: A large-scale hierarchical image database'. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, pp. 248–255.

Dugas, Charles et al. (2001). 'Incorporating second-order functional knowledge for better option pricing'. In: *Advances in Neural Information Processing Systems*, pp. 472–478.

Efron, Bradley (1986). 'Why isn't everyone a Bayesian?' In: *The American Statistician* 40.1, pp. 1–5.

Fawzi, Alhussein, Seyed-Mohsen Moosavi-Dezfooli and Pascal Frossard (2016). 'Robustness of classifiers: from adversarial to random noise'. In: *Advances in Neural Information Processing Systems*, pp. 1624–1632.

Fukushima, Kunihiko (1975). 'Cognitron: A self-organizing multilayered neural network'. In: *Biological cybernetics* 20.3-4, pp. 121–136.

— (1988). 'Neocognitron: A hierarchical neural network capable of visual pattern recognition'. In: *Neural networks* 1.2, pp. 119–130.

Gelman, Andrew et al. (2008). 'Objections to Bayesian statistics'. In: *Bayesian Analysis* 3.3, pp. 445–449.

Glorot, Xavier, Antoine Bordes and Yoshua Bengio (2011). 'Deep sparse rectifier neural networks'. In: *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* 15, pp. 315–323. ISSN: 15324435. DOI: 10.1.1.208.6449. arXiv: 1502.03167.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Goodfellow, Ian J, Jonathon Shlens and Christian Szegedy (2015). 'Explaining and Harnessing Adversarial Examples'. In: *Iclr 2015*, pp. 1–11.

Goodfellow, Ij, J Pouget-Abadie and Mehdi Mirza (2014). 'Generative Adversarial Networks'. In: *arXiv preprint arXiv: ...* pp. 1–9. ISSN: 10495258. arXiv: arXiv:1406.2661v1. URL: http://arxiv.org/abs/1406.2661.

Gu, Jiuxiang et al. (2015). 'Recent Advances in Convolutional Neural Networks'. In: *arXiv preprint arXiv:1512.07108*.

Gu, Shixiang and Luca Rigazio (2014). 'Towards Deep Neural Network Architectures Robust to Adversarial Examples'. In: *NIPS Workshop on Deep Learning and Representation Learning* 2013, pp. 1–9. DOI: 10.5121/ijcsa.2014.4310.

He, Kaiming et al. (2016). 'Deep residual learning for image recognition'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). 'Long short-term memory'. In: *Neural computation* 9.8, pp. 1735–1780.

Hornik, Kurt (1991). 'Approximation capabilities of multilayer feedforward networks'. In: *Neural Networks* 4.2, pp. 251–257. ISSN: 08936080. DOI: 10.1016/0893-6080(91)90009-T.

Hubel, D H and T N Wiesel (1968). 'Receptive fields and functional architecture of monkey striate cortex.' In: *The Journal of physiology* 195.1, pp. 215–43. ISSN: 0022-3751.

Ian J. Goodfellow et al. (2013). 'Maxout networks'. In: URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.421.9321.

Ioffe, Sergey and Christian Szegedy (2015). 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *Journal of Machine Learning Research*. DOI: 10.1007/s13398-014-0173-7.2.

Jia, Yangqing et al. (2014). 'Caffe: Convolutional architecture for fast feature embedding'. In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, pp. 675–678.

Kingma, Diederik and Jimmy Ba (2014). 'Adam: A method for stochastic optimization'. In: *arXiv preprint arXiv:1412.6980*.

Krizhevsky, Alex and Geoffrey Hinton (2009). 'Learning multiple layers of features from tiny images'. In:

Krizhevsky, Alex, IIya Sulskever and Geoffret E Hinton (2012a). 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information and Processing Systems (NIPS)*, pp. 1–9.

— (2012b). 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information and Processing Systems (NIPS)*, pp. 1–9.

Kurakin, Alexey, Ian Goodfellow and Samy Bengio (2016). 'Adversarial examples in the physical world'. In: *Arxiv*, pp. 1–15. arXiv: 1607.02533. URL: http://arxiv.org/abs/1607.02533.

LeCun, Yann et al. (1998a). 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Yann, Corinna Cortes and Christopher J.C. Burges (1998b). *The MNIST database of handwritten digits*. URL: http://yann.lecun.com/exdb/mnist/ (visited on 17/04/2017).

Lecun, Yann, Yoshua Bengio and Geoffrey Hinton (2015). 'Deep learning'. In: *Nature* 521.7553, pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539. arXiv: arXiv:1312.6184v5.

Lichman, M. (2013). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Lin, Min, Qiang Chen and Shuicheng Yan (2013). 'Network in network'. In: *arXiv preprint arXiv:1312.4400*.

Lou, Yan et al. (2016). 'Foveation-based Mechanisms Alleviate Adversarial Examples'. In:

Maharaj, Akash V (2016). 'Improving the adversarial robustness of ConvNets by reduction of input dimensionality'. In:

Mahendran, Aravindh and Andrea Vedaldi (2015). 'Understanding deep image representations by inverting them'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5188–5196.

Min, Seonwoo, Byunghan Lee and Sungroh Yoon (2016). 'Deep Learning in Bioinformatics'. In: ISSN: 1467-5463. DOI: 10.1093/bib/bbw068. arXiv: 1603.06430. URL: http://arxiv.org/abs/1603.06430.

Miyato, Takeru et al. (2015). 'Distributional smoothing by virtual adversarial examples'. In: *ICLR*.

Mnih, Volodymyr et al. (2015). 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540, pp. 529–533.

Moore, Gordon E et al. (1998). 'Cramming more components onto integrated circuits'. In: *Proceedings of the IEEE* 86.1, pp. 82–85.
Moosavi-Dezfooli, Seyed-Mohsen et al. (2016). 'Universal adversarial perturbations'. In: *arXiv preprint arXiv:1610.08401*.
Murphy, Kevin P. (2012). *Machine Learning: a Probabilistic Perspective*. MIT Press.
Murty, Katta G. and Santosh N. Kabadi (1987). 'Some NP-complete problems in quadratic and nonlinear programming'. In: *Math. Program.* 39.2, pp. 117–129. DOI: 10.1007/BF02592948.
Newell, Allen, John C Shaw and Herbert A Simon (1959). 'Report on a general problem solving program'. In: *IFIP congress*. Vol. 256. Pittsburgh, PA, p. 64.
Papernot, Nicolas et al. (2016). 'Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks'. In: *IEEE Symposium on Security and Privacy*, pp. 1–16. DOI: 10.1109/SP.2016.41. eprint: 1511.04508.
Ribeiro, Marco Tulio, Sameer Singh and Carlos Guestrin (2016). '"Why Should I Trust You?": Explaining the Predictions of Any Classifier'. In: *arXiv*. ISSN: 9781450321389. DOI: 10.1145/1235. arXiv: 1602.04938. URL: http://arxiv.org/abs/1602.04938.
Rosenblatt, Frank (1958). 'The perceptron: A probabilistic model for information storage and organization in the brain.' In: *Psychological review* 65.6, p. 386.
Rozsa, Andras, Ethan M Rudd and Terrance E Boult (2016a). 'Adversarial Diversity and Hard Positive Generation'. In: *arXiv preprint arXiv:1605.01775*.
Rozsa, Andras, Manuel Gunther and Terrance E Boult (2016b). 'Towards Robust Deep Neural Networks with BANG'. In: *arXiv preprint arXiv:1612.00138*.
Russakovsky, Olga et al. (2015). 'ImageNet Large Scale Visual Recognition Challenge'. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.
Simonyan, Karen and Andrew Zisserman (2015). 'Very deep convolutional networks for large-scale image recognition'. In: *Iclr*. DOI: 10.1016/j.infsof.2008.09.005.
Spivak, Michael (1994). *Calculus*. 3rd ed. Houston, TX: Publish or Perish.
Srivastava, Nitish et al. (2014). 'Dropout : A Simple Way to Prevent Neural Networks from Overfitting'. In: *Journal of Machine Learning Research (JMLR)* 15, pp. 1929–1958. ISSN: 15337928. DOI: 10.1214/12-AOS1000. arXiv: 1102.4807.
Szegedy, Christian, W Zaremba and I Sutskever (2013). 'Intriguing properties of neural networks'. In: *arXiv preprint*, pp. 1–10. ISSN: 15499618. DOI: 10.1021/ct2009208. arXiv: arXiv:1312.6199v4. URL: http://arxiv.org/abs/1312.6199.
Szegedy, Christian et al. (2015). 'Going deeper with convolutions'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.
Warde-Farley, David and Ian Goodfellow (2016). 'Adversarial Perturbations of Deep Neural Networks'. In: *Perturbation, Optimization and Statistics*. Ed. by Tamir Hazan, George Papandreou and Daniel Tarlow. MIT Press. Chap. 1.
Winograd, Terry (1972). 'Understanding natural language'. In: *Cognitive psychology* 3.1, pp. 1–191.
Xia, Fei and Ruishan Liu (2016). 'Adversarial Examples Generation and Defense Based on Generative Adversarial Network'. In:
You, Quanzeng et al. (2016). 'Image Captioning With Semantic Attention'. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
Zhang, Chiyuan et al. (2016). 'Understanding deep learning requires rethinking generalization'. In: *arXiv preprint arXiv:1611.03530*.
Zheng, Stephan et al. (2016). 'Improving the robustness of deep neural networks via stability training'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4480–4488.