UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Stochastic Relaxation of Deep Neural Networks as a Way to Build Adversarial Robustness

Author:
Solomiia LENO

Supervisor:
Dr. Boris FLACH

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

I, Solomiia LENO, declare that this thesis titled, "Stochastic Relaxation of Deep Neural Networks as a Way to Build Adversarial Robustness" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Science is like sex: sometimes something useful comes out, but that is not the reason we are doing it."

Richard P. Feynman

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Stochastic Relaxation of Deep Neural Networks as a Way to Build Adversarial Robustness

by Solomiia LENO

Abstract

Deep Neural Networks show spectacular results on real-life tasks from different applications: recommendation systems, speech recognition, autonomous driving, *etc.* Despite this success they were proven to be vulnerable to small perturbations, imperceptible to human eye, in input data called adversarial attacks. Main reasons of such vulnerability lie in the overparametrization of DNNs, tendency to overfitting and high variance of learned features.

In this work we show that stochastic relaxation of Deep Neural Networks impacts those factors and can help to improve adversarial robustness of a model up to $\times 1.7$ times. We perform experiments on Binary and ReLU Convolutional Neural Networks and later compare our method results with current SOTA approach to building adversarial robustness - adversarial learning. In conclusions we propose steps that might be taken to further improve performance of Stochastic Neural Networks on both clean and adversarial data.

Acknowledgements

I want to express deep gratitude to my supervisor, prof. Boris Flach, head of Machine Leaning group at Czech Technical University in Prague, for helping me with this research, sharing his experience and knowledge, and helping me out with every single issue whether it was a dump coding mistake or a complicated mathematical question.
I also want to say big thanks to every lecturer, lab teacher and teaching assistant who I've met in Ukrainian Catholic University and Czech Technical University during past four years for sharing their knowledge and inspiring me to always do even more.
I am also super grateful to my family and closest friends for their constant support and belief in me.
Last but definitely not least I want to express the deepest gratitude to The Armed Forces of Ukraine for making it possible for me to do this research and write my thesis under a peaceful sky.

Contents

List of Figures

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
DL	**D**eep **L**earning
DNN	**D**eep **N**eural **N**etwork
FFNN	**F**eed **F**orward **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
BNN	**B**inary **N**eural **N**etwork
ReLU	**Re**ctified **L**inear **U**nit
CE	**C**ross-**E**ntropy
NLL	**N**egative **L**og-**L**ikelihood
MLE	**M**aximum **L**ikelihood **E**stimator
FGSM	**F**ast **G**radient **S**ign **M**ethod
SOTA	**S**tate-**O**f-**T**he-**Art**

Chapter 1

Introduction

1.1 Deep Learning in Real-life Applications

Deep Neural Networks (DNNs) have become the leading pattern recognition technology in modern artificial intelligence. They have proved incredibly successful at correctly classifying all kinds of input, including images, speech, data on consumer preferences and others. Moreover, they have been shown to successfully analyze and extract useful knowledge from both large amounts of data and non homogeneous data collections, *i.e.,* data collected from different sources [1]. DNNs have become a part of daily life, running almost everything from automated telephone systems to user recommendations on the streaming service Netflix. The main reason for such wide popularity of DNNs today are drastically increased chip processing abilities (*e.g.,* GPU units), the significantly low cost of computing hardware, and recent advances in machine learning and signal/information processing research [37].

1.2 Downfalls of Deep Learning

While showing outstanding results in various real-life tasks, Deep Learning algorithms have also raised a widely shared concerns and mixed sentiments among AI researchers. In 2013 Christian Szegedy *et al.* published a paper called *"Intriguing properties of neural networks"* [38] where they showed that it is possible to take one image - of a lion, for example - and by altering few pixels convince a DL model that it is actually looking at something different, like a panda. Such doctored examples were later called adversarial examples and started a new branch of research on how to protect DL models from *adversarial attacks*. And while misclassifying an animal specie doesn't seem to deserve that much attention of a scientific community here is another example. A year later, in 2014 a group of researchers showed how one can fool an AI-based autonomous driver into misreading a "stop" sign as "speed limit 45" sign by carefully putting few white stickers on the sign [10]. Later research shows that such an issue goes beyond object recognition: Long-Short Term Memory models for speech recognition [45], Reinforcement Learning models for game playing [20] and other types of Deep Learning models can also be attacked with slight changes to input data.

1.3 Existing Methods to Deter Adversarial Attacks on DNNs

Adversarial attacks are essentially small (*i.e.,* imperceptible to the human eye) perturbations of an input data which cause the model to misclassify an example. The key insight is that those small perturbations of input values cause larger perturbations of network activations that result in incorrect predictions. Thus, there exist two

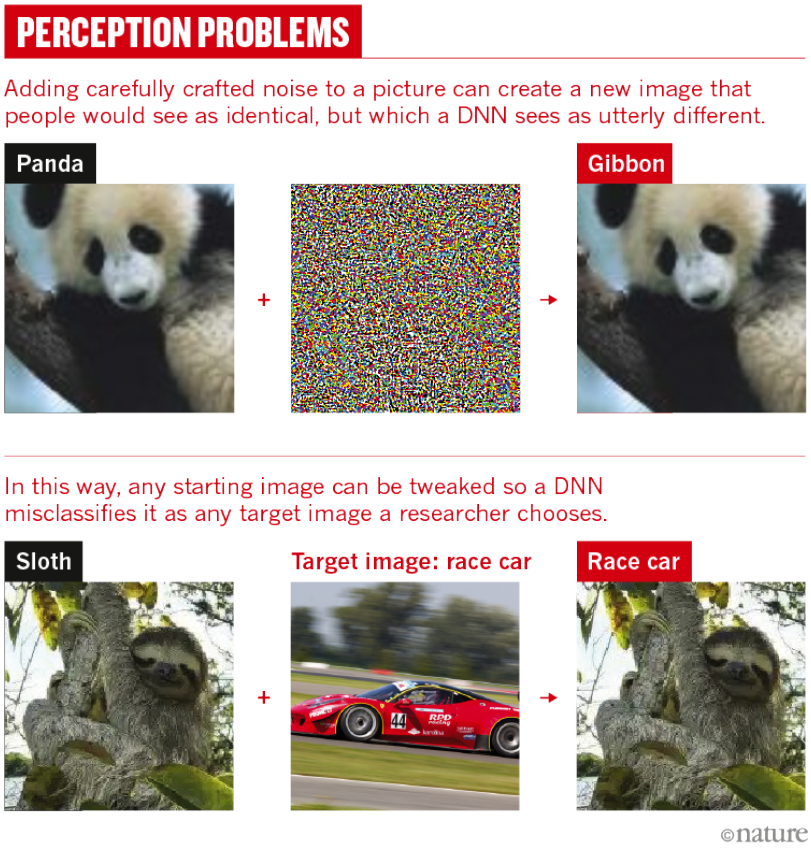

FIGURE 1.1: Adversarial examples: *top* - untargeted adversarial example, *i.e.* the attacker tries to misguide the model to predict any of the incorrect classes, *bottom* - targeted adversarial example, *i.e.* misguiding the model to a particular class other than the true class [32]. Image source: [16]

main approaches to building adversarial robustness: a) regularization of input data and/or model parameters to prevent small changes in data from resulting in large changes in representation vectors, b) training model with adversarial examples. Research on this issue is still quite active but currently the most popular methods of adversarial attacks deterrence are the following ones:

- **Adversarial learning**. SOTA approach to building adversarial robustness, originally proposed by Szegedy *et al.* in 2014 [38] adversarial learning provided DL model training on a mixture of clean data and adversarial examples created by researchers and/or simple algorithms. Later this approach was advanced to include two DL models: one for solving original task and another for generating adversarial examples during the training process [15].

- **Input gradient regularization**. Input data regularization techniques are useful to avoid large gradients on the input that make model vulnerable to attacks. This idea was proven to be competitive with adversarial learning both theoretically and empirically [12].

- **Feature squeezing**. These methods reduce the search space available to an adversary by coalescing samples that correspond to many different feature vectors in the original space into a single sample [43].

- **Contrastive learning**. This machine learning technique is essentially unsupervised learning with supervised fine-tuning [11]. Key idea behind this method is that unsupervised learning methods learn better representations which are initially more resistant to adversarial examples.

1.4 Focus of the Work

In this work we focus on the stochastic relaxation of Deep Neural Networks as one of the ways to regularize distribution of layers activations and possibly improve model robustness against adversarial attacks. This method also requires almost no additional memory and/or computational power usage. Given a large variety of tasks that are being solved with DL algorithms today, we stop at one of easiest - image classification. Relative simplicity of classification models architecture will allow us to experiment with both different models of stochastic neurons and most popular Neural Networks regularization techniques.

Chapter 2

Background

2.1 Artificial Neural Networks

Artificial Neural Networks are computational systems inspired by the structure of a human brain. They consist of a sequence of connected layers of artificial neurons that can transmit input signal to the output of the network. An artificial neuron is essentially a linear function that maps all input signals to a scalar which is then passed as one of the inputs to the next layer. Since composition of linear functions is a linear function itself, an activation function is used after each neuron to introduce non-linearity to the model. Due to presence of non-linearities and multi-layer architecture, ANNs are particularly useful to extract non-trivial patterns, especially when the problem is too hard to solve with handcrafted features. They are typically trained by optimizing a loss function - an error between the ground-truth and predicted output for the given data input.
Feed Forward Network is the simplest and the most widely used type of ANN, where the information moves only in one direction. Architecture of FFNN can essentially be represented with an acyclic graph as on Figure 2.1. The training of FFNN is generally done via the backpropagation algorithm, which efficiently calculates the gradient in the weights w.r.t. a loss function and updates the parameters in order to minimize the prediction errors.

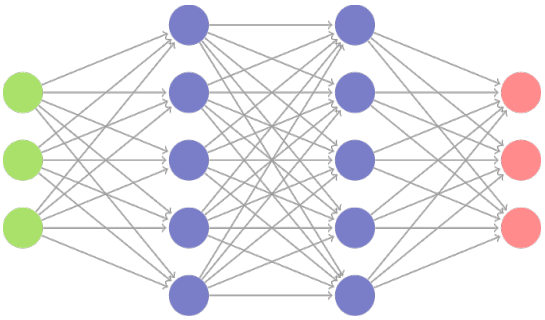

FIGURE 2.1: Example of Feed Forward Neural Network architecture: in *green* - input layer, in *purple* - two hidden layers, in *red* - output layer which is usually followed by a loss function.

2.2 Convolutional Neural Networks

Convolutional Neural Networks [25] is a specialized kind of neural networks for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a $1D$ grid taking samples at regular time intervals, and image data, which can be thought of as a $2D$ grid of pixels. From architecture

and training point of view CNNs do not differ from ANNs: they also consist of a sequence of linear layers followed by non-linear activation functions and are trained with backpropagation. The only difference is that instead of dense matrix multiplication in every layer one performs convolution operation, which is application of sliding kernel to the input data.
Given $w \in \mathbb{R}^{2h+1}$ is a kernel of weights, $x \in \mathbb{R}^n$ is an input data vector and $y \in \mathbb{R}^m$, where $m = n - 2h$, is a kernel output, convolution in $1D$ is defined as following:

$$y = w \star x : y_i = \sum_{j=-h}^{h} w_i x_j \tag{2.1}$$

Such operation is useful due to high correlation of data points with the neighbouring ones. Besides efficient utilisation the data structure, convolution brings few more advantages to CNNs:

- **Reduction of number of learnable parameters**. Kernel of a fixed size (usually $h = 1$ or $h = 3$) is significantly smaller than the size of an input data.
- **Translation equivariance**. If an object in input data is shifted, the output of convolution will be shifted likewise.

It was a common practice to combine convolution layers with pooling (*e.g.,* max pooling) layers to introduce dimensionality reduction to the model. However, due to generalization of convolution with stride (step size of the kernel when traversing the input data) and dilation (spacing between values in a kernel) parameters pooling layers are considered unnecessary.
Similarly to ANNS, construction of CNNs is closely related to the way human brain processes visual information. The main similarities are found in low-lever features extraction (illustrated in Fig. 2.2) performed by the primary visual cortex (*a.k.a.* V1) cells in human brain and first few convolutional layers of CNNS.

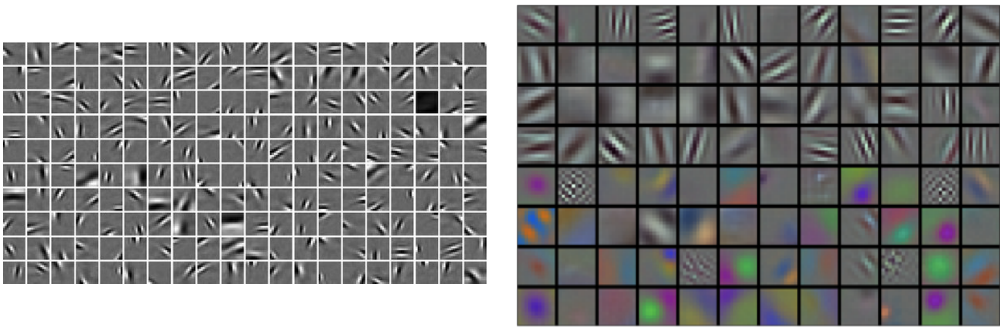

FIGURE 2.2: On *left* - examples of Gabor function - mathematical model describing the cells' weights in V1 [29] - output on various parameters, on *right* - low-level kernels learned on the images from ImageNet by AlexNet model [23].

2.3 Stochastic Neural Networks

It is well known from Statistical Pattern Recognition [27] that stochastic predictors can not be better than their deterministic counterparts. Using deterministic models together with standard loss functions (*e.g.,* $0/1$ loss), on the other hand, makes

the corresponding empirical loss to be a piece-wise constant function. The common solution is to interpret the network outputs as class probabilities and use Maximum Likelihood Estimator (Cross Entropy) instead. Stochastic Neural Networks, however, allow one to interpret the network outputs as probabilities of a stochastic predictor (*i.e.,* to directly model the probability distribution) and optimize model parameters by minimizing corresponding expected loss.
There are two main ways to introduce stochasticity to a Neural Network: a) use stochastic parameters, b) use stochastic activation functions. In this work we use the second approach, *i.e.,* we introduce additive or multiplicative noise to the neuron pre-activation. For a neuron with additive noise its output is defined as following:

$$a = Wx + b \tag{2.2}$$

$$y = f(a - \xi) \tag{2.3}$$

where $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$ are neuron weight and bias parameters respectively, a is neuron pre-activation, $f(\cdot)$ is an activation function and ξ is (a vectors of) *independent noise* with a simple probability distribution (*e.g.,* Gaussian, logistic).
Similarly, an artificial neuron with multiplicative noise is defined as:

$$y = f(a \cdot \xi) \tag{2.4}$$

where ξ is (a vector of) independent noises of same dimensionality as pre-activations a and $\cdot$ is an operation of element-wise multiplication.
In this stochastic relaxation the expected loss becomes a smooth function and, thus, is differentiable in network parameters, however it raises a problem of finding a suitable (preferably unbiased) gradient estimator for backpropagation.

2.4 Gradient Estimators

Learning parameters of an Artificial Neural Network is a task of continuous optimization which is usually solved with gradient-based methods (*e.g.,* Stochastic Gradient Descent). Similarly learning parameters of a Stochastic Neural Network is a task of *stochastic* optimization, where instead of explicitly computing gradient of expectation of objective function w.r.t. function arguments one has to find or derive a stochastic gradient estimator.
Task of learning parameters of a Stochastic Neural Network is defined as following:

$$\mathbb{E}_{x \sim p(x)}[l(x, \theta)] \to \max_{\theta} \tag{2.5}$$

where x is a vector of input examples, θ is a vector of network parameters and l is a loss function. The gradient w.r.t. network parameters θ is then:

$$\frac{d}{d\theta}\mathbb{E}_x[l(x, \theta)] = \mathbb{E}_x[\frac{d}{d\theta}l(x, \theta)] = \mathbb{E}_x[\frac{d}{dy^L}l(x, \theta)\frac{d}{d\theta}y^L] \tag{2.6}$$

where L is the number of network layers and, thus, y^L is the output of the last network layer. Typical approach to calculating such an expression is to sample data from $p(x)$ and to approximate the theoretical expectation with an empirical expectation, however nature of some widely used activation functions (*e.g.,* sign function which is not differentiable in 0 and has zero derivative everywhere else) might pose additional challenges.

2.4.1 Straight-Through Gradient Estimator

Straight-Through estimator is a **biased** gradient estimator commonly used in Quantized Neural Networks. The idea of the straight-through estimator is to simply bypass the piece-wise constant activation in the backward pass and treat it as if it was an identity function, hence the name. This ad-hoc solution was first introduced empirically, so now there exist few other options that were also derived for some specific practical cases. In this work, we use slightly different definition of STE, which has a theoretical justification [36], to approximate gradients in the Stochastic Binary Networks.
We define a stochastic binary neuron as a noisy sign mapping:

$$y = \text{sign}(a - z) \tag{2.7}$$

where $a = Wx + b$ is neuron pre-activation and z is a vector of independent noises. Equivalently one can say that y is a random variable that follows $\{-\mathbf{1}, \mathbf{1}\}$ valued Bernoulli distribution with probability $p(y = 1) = \mathbb{P}(a - z > 0) = \mathbb{P}(a > z) = F(a)$ where F denotes the c.d.f. of noise distribution. Due to definition of a Bernoulli r.v., $\mathbb{E}[y] = p(y = 1) = F(a)$.
Now one can define a straight-through gradient estimator for the binary neuron as following:

$$\frac{d}{d\theta}\mathbb{E}_{x,z}[l(x,\theta)] = \mathbb{E}_{x,z}[\frac{d}{dy^L}l(x,\theta)\frac{d}{d\theta}y^L] = \mathbb{E}_x[\frac{d}{dy^L}l(x,\theta)\frac{d}{d\theta}\mathbb{E}_z y^L] \tag{2.8}$$

For $z \sim logistic$, $p(y = 1) = F_{log}(a) = \sigma(a)$ where $\sigma(\cdot)$ is a sigmoid function. Thus, equation 2.8 can be finished as:

$$\mathbb{E}_x[\frac{d}{dy^L}l(x,\theta)\frac{d}{d\theta}\mathbb{E}_z y] = \mathbb{E}_x[\frac{d}{dy^L}l(x,\theta)\frac{d}{d\theta}\sigma(a)] \tag{2.9}$$

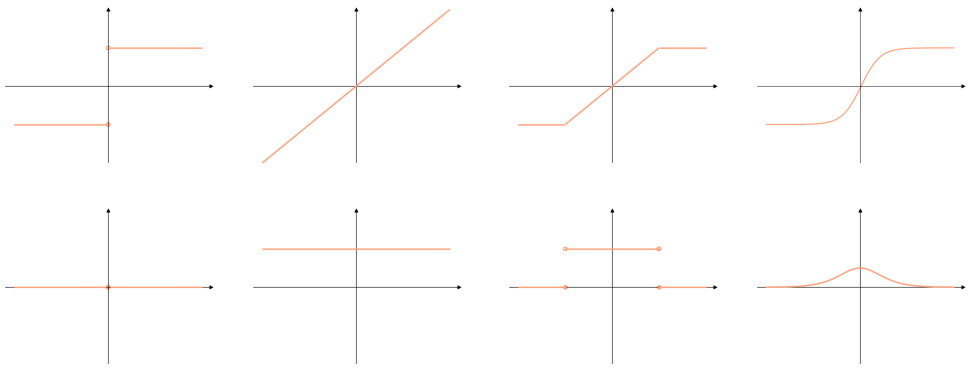

FIGURE 2.3: The sign function and proxy functions for derivative used in the STE. *First* column - sign function and its derivative, *second* column - identity and function and its derivative, originally used in the STE, *third* - hard tanh function which becomes a proxy for step function when uniform noise (*below*) is injected to the neuron, *last* column - tanh function which becomes the proxy is one injects logistic noise illustrated *below*.

2.4.2 Score Function Gradient Estimators

The Score Function Gradient Estimator is one the easiest and most general **unbiased** estimators which is widely used in research and is also known as the likelihood ratio method [13] and REINFORCE estimator [40].
The *score function* itself is the derivative of log of a function:

$$\frac{d}{d\theta} f(x,\theta) = f(x,\theta)\frac{d}{d\theta}\log f(x,\theta) \tag{2.10}$$

This identity is useful for deriving a gradient estimator with a Monte Carlo method, *i.e.*, sampling data from distribution and calculating empirical expectation:

$$\begin{aligned}\frac{d}{d\theta}\mathbb{E}_{x\sim p(x,\theta)} f(x) &= \frac{d}{d\theta}\int p(\mathbf{x},\theta) f(\mathbf{x}) d\mathbf{x} = \int \frac{d}{d\theta} p(\mathbf{x},\theta) f(\mathbf{x}) d\mathbf{x} = \\ &= \int p(\mathbf{x},\theta)\frac{d}{d\theta}[\log p(\mathbf{x},\theta)] f(x) d\mathbf{x} = \mathbb{E}_x[f(x)\frac{d}{d\theta}\log p(x,\theta)] \\ &\sim \frac{1}{N}\sum_{i=1}^{N}[f(x_i)\frac{d}{d\theta}\log p(x_i,\theta)]\end{aligned} \tag{2.11}$$

It is worth to note that the Score Function Estimator has no restrictions on the form of $f(\cdot,\theta)$, its continuity and differentiability. Another important comment is that while being unbiased, this estimator has a **high variance**.

Chapter 3

Related works

3.1 Studies on Properties of Stochastic Neural Networks

It is well known that Multilinear Feed Forward Neural Networks are the universal approximators for the real-valued functions [19]. However, FFNNs are capable only of directly mapping a set of inputs to a set of outputs which is not enough for modeling stochastic processes. Thus, in 2004 Stochastic Neural Networks were introduced, initially as a mathematical tool for modeling biological and physical processes [7]. As mentioned in section 2.3, one can also interpret SNNs outputs as probabilities of a stochastic predictor. Due to variety of interpretations and ways to introduce stochasticity to a Neural Network, training of the SNNs has been a wide and important topic of research for many years [26, 30].
The later research was more focused on generalizing properties of SNNs. Following the studies on approximating abilities of ANNs, both deep and shallow stochastic sigmoid feedforward networks were shown to be capable of universal approximation [28]. Combination of ability to model complicated probabilistic distributions and universal approximation suggested that SNNs might also have generative abilities. Despite thorough research they did not outperform Generative Adversarial Networks but showed to be useful for real-life applications.
Stochastic Neural Networks have found applications not only as generative models, but also in unsupervised feature learning [17, 31], semantic hashing [33], and natural language understanding [35], among others. Unsupervised training with SNNs can be used as a parameter initialization strategy for subsequent supervised learning, which was a key technique in the rise of Deep Learning in the years 2000s [18, 3].

3.2 Network Regularizations for Increasing Adversarial Robustness

Building adversarially robust models is an optimization problem with two objectives: (i) maintain test accuracy on clean unperturbed images, and (ii) be robust to large adversarial perturbations. In chapter 1 we briefly discuss main approaches for solving such a problem. Current SOTA approaches which are mostly variations of adversarial learning with heuristics usually improve adversarial robustness at a cost of test accuracy. Moreover they usually are vulnerable to new, stronger attacks [6]. This had lead research community to develop theoretical tool to *certify* adversarial robustness.
First theoretical approaches were based on linear and mixed-linear programming [41]. These methods are what is are called *provably robust*, *i.e.*, they can verifiably guarantee that for a given data example x, no perturbation Δ in l_∞ less that some

specified ϵ can change the class label that the network predicts for the perturbed example $x + \Delta$. However such provable rosubtness can be guaranteed only for rather small-sized networks, moreover general effectiveness of these approaches does not scale well to deeper networks.
The later approaches were based on randomized smoothing [34] or estimates of the local Lipshitz constant. Randomized smoothing of a classifier $f : \mathbb{R}^d \to \mathcal{Y}$ returns a smoothed classifier g which for input image x predicts class label that is most likely to be returned by base classifier f when x is perturbed by isotropic Gaussian noise:

$$g(x) = \arg\max_{c \in \mathcal{Y}} \mathbb{P}(f(x + \epsilon) = c) \tag{3.1}$$

where $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$. An equivalent definition is that $g(x)$ predicts the class c whose pre-image $\{x' \in \mathbb{R}^d : f(x') = c\}$ has the largest probability measure under the distribution $\mathcal{N}(\mu, \sigma^2)$. The noise level σ is a hyperparameter of the smoothed classifier $g(\cdot)$ which controls a robustness/accuracy tradeoff; it does not change with the input x.
Randomized smoothing method scaled well to Deep Networks trained on ImageNet-1k. It is also worth mentioning that randomized smoothing combined with adversarial training has shown great promise for adversarial robustness certification against attacks in the l_2-norm , although it is not yet clear how randomized smoothing may be adapted to other norms.
In 2019 Finlay and Oberman [12] derived a theoretical lower bound for the perturbation to change model decision. This derivation was based on assumption of Lipschitz continuity of a loss function and motivated a need for gradient regularization which lead to increase of a minimum adversarial distance - minimum distance between clean data sample and an adversarial one - for which adversarial data sample is classified correctly. Gradient regularization method proved to be competitive with adversarial training and scaled well to ImageNet-1k.

Chapter 4

Methodology

4.1 Scope of Work

In this work, we perform experiments with Stochastic Convolutional Neural Networks with sign and ReLU activation functions. To implement the stochastic relaxation of the networks we inject each block - Conv2D + *(optionally)* BatchNorm2D + activation function - with either additive or multiplicative noise before applying activation function, *i.e.*, we use stochastic activations. We also want to try different models of stochastic neurons, thus we use two different noise distributions: Gaussian and logistic.

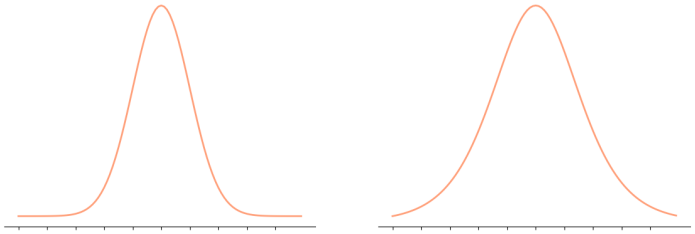

FIGURE 4.1: Probability density functions of chosen noise distributions: Gaussian (std = 2) on left and logistic (std = 2) on right

For gradient estimation in binary stochastic neurons we use Straight-Trough and Score Function Gradient Estimators discussed in section 2.4. ReLU activation is everywhere differentiable, so backpropagation in the Stochastic ReLU Networks poses no additional challenges for us.
Measuring and analysing model performance on adversarial examples is closely related to the analysis of generalization abilities of the model. Thus, besides measuring model performance on adversarial attacks we also want to perform qualitative experiments to see how the injected noises affect model accuracy, model certainty on both correct and wrong predictions, explore extracted fearures *etc.* In the conclusions, we compare performance of stochastic models with the SOTA approach which is training model on a mixture of clean and adversarial data.

4.2 Stochastic Artificial Neuron

In section 4.1 we mentioned that we introduce stochasticity to networks by either adding or multiplying neuron pre-activation by an independent noise sampled from *Gaussian* and *logistic* distributions. We are still left with the choice of noise distribution parameters.

Expectation and variance of sum and product of two independent random variables X and Z is defined as following:

$$\mathbb{E}[X + Z] = \mathbb{E}[X] + \mathbb{E}[Z] \tag{4.1}$$

$$\mathbb{V}\text{ar}[X + Z] = \mathbb{V}\text{ar}[X] + \mathbb{V}\text{ar}[Z] \tag{4.2}$$

$$\mathbb{E}[XZ] = \mathbb{E}[X]\mathbb{E}[Z] \tag{4.3}$$

$$\mathbb{V}\text{ar}[XZ] = \mathbb{V}\text{ar}[X]\mathbb{V}\text{ar}[Z] + \mathbb{V}\text{ar}[X]\mathbb{E}^2[X] + \mathbb{V}\text{ar}[Z]\mathbb{E}^2[Z] \tag{4.4}$$

From these definitions one might have two proposals:

- set $\mathbb{E}[Z] = 0$ and $\mathbb{V}\text{ar}[Z] = 1$ in order to preserve statistics of distribution of layer pre-activations. In this approach multiplicative noise can be viewed as a random reweighting of the features extracted by the layer which might result in more uniform distribution of feature importance.
- use noise addition/multiplication as a regularization of pre-activation distribution, *i.e.,* set $\mathbb{E}(Z)$ and $\mathbb{V}\text{ar}(Z)$ to some data-driven values.

In this work we explore the noise distributions with $\mathbb{E}(Z) = 0$ to preserve asymptotic unbiasedness of the predictors. However, we change the value of $\mathbb{V}\text{ar}(Z)$, or equivalently $\text{std}(Z)$, throughout the experiments. We treat the ratio between standard deviation of the clean data distribution and the standard deviation of the noise distribution as a *noisiness ratio* - the level of *noisiness* injected in the network.
Namely, the statistics of the clean dataset are $\mathbb{E}(X) = 0.5$ and $\text{std}(X) = 0.5$ (these statistic were calculated on random sample of 7000 images, which is 10% of original dataset, drawn from both training and test parts of the dataset). Thus, to achieve 5%, 10% and 20%-level of *noisiness* we sample noises from distributions with $\text{std}(Z) = [0.025, 0.05, 0.01]$.

4.3 Adversarial Attacks

4.3.1 Targeted Adversarial Attack

The traditional strategy for finding a targeted adversarial example is as follows: given some classifier $p(y|x)$, some input $x \in \mathbb{R}^n$, some target class $\hat{y}$ and a maximum perturbation ϵ, we want to find the input $\hat{x}$ that maximizes $p(\hat{y}|\hat{x})$ (or equivalently $\log p(\hat{y}|\hat{x})$), subject to the constraint $||x - \hat{x}||_\infty \leq \epsilon$. When $p(y|x)$ is parameterized by a neural network, an attacker with access to the model can perform iterative gradient descent on x in order to find a suitable input $\hat{x}$. Such an approach was first introduced in 2016 as *Iterative Fast Gradient Sign Method* [24] and is mathematically defined as following:

$$x_0^{adv} = x, \qquad x_i^{adv} = \text{clip}_{0,max(x)-\epsilon}(x_{i-1}^{adv} - \epsilon\ \text{sign}(\nabla_x J(\theta, x_{i-1}^{adv}, y_{LL}))) \tag{4.5}$$

where clip function is used to keep the overall change in the ϵ-range, ϵ is the maximum perturbation hyperparameter, J is an objective function used for model training, θ is a vector of model parameters and y_{LL} is a target class, the one which initially is considered the least likely for a data sample x. Instead of the least likely class as a target one can use any class other that the ground-truth one.

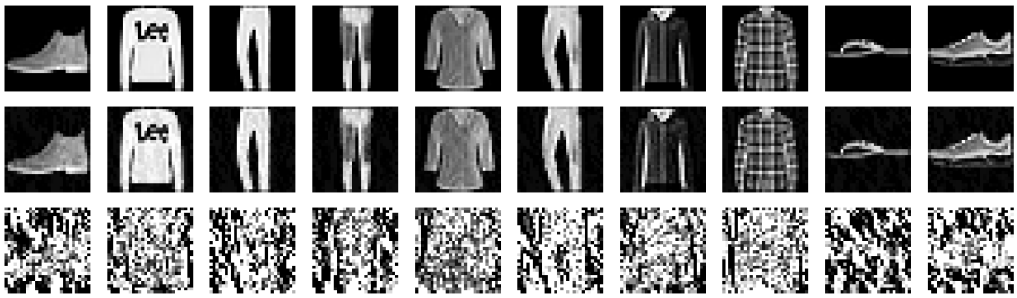

FIGURE 4.2: Targeted adversarial examples generated with the Iterative Fast Gradient Sign Method. *Top* row - original images from Fashion-MNIST dataset, values are scaled to $[0, 1]$ range, *middle* row - adversarial examples generated with FGSM, maximum perturbation $\epsilon = 0.05$, also in $[0, 1]$ value range, *bottom* row - absolute differences between original and adversarial images, value range $[0, 0.05]$.

4.3.2 Untargeted Adversarial Attack

Untargeted adversarial attacks are considered weaker than targeted ones in a sense that they usually provide less change to the input image but at the same time they are easier and computationally cheaper to implement.
For data sample x, respective ground-truth label y, model with parameters θ and loss function J an untargeted one-iteration variant of the Fast Gradient Sign Method is defined as following:

$$x^{adv} = x + \epsilon \operatorname{sign}(\nabla_x J(\theta, x, y)) \tag{4.6}$$

From such a definition one cas see that this method relies on increasing the loss value associated with the ground-truth label hoping to "push" loss for other classes lower. However, this might not be the case: untargeted FGSM may return an adversarial example which is still correctly classified by the model or is classified as something very similar (*e.g., boot* classified as a *sneaker*).

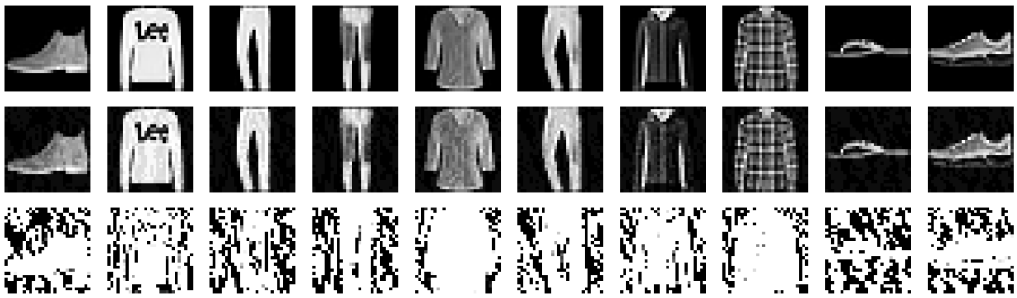

FIGURE 4.3: Untargeted adversarial examples generated with Fast Gradient Sign Method. *Top* row - original images from Fashion-MNIST dataset, values are scaled to $[0, 1]$ range, *middle* row - adversarial examples generated with FGSM, maximum perturbation $\epsilon = 0.05$, also in $[0, 1]$ value range, *bottom* row - absolute differences between original and adversarial images, value range $[0, 0.05]$.

4.3.3 Missing Features Attack

We introduce one more metric for model robustness estimation - missing features attack. Motivation for such a metric comes from the observations of heat maps produced by layers activation for certain data examples (illustrated on Fig.4.4): some images from different classes cause very similar activation of later network layers leading us to believe that not all class-defining features are taken by network into account. We observed such a behaviour in both Binary and ReLU baseline models, moreover it happens not only for classes of similar articles of clothing (*e.g., shirt* and *coat*) but also for dissimilar ones (*e.g., boot* and *bag*).

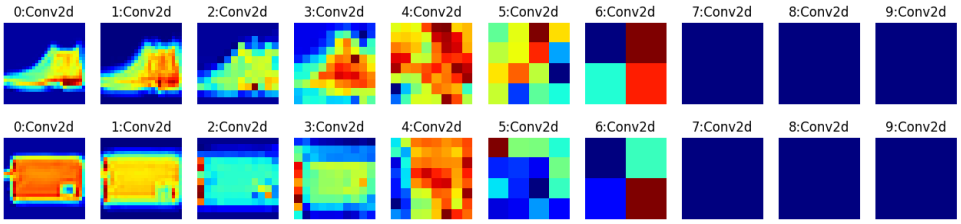

FIGURE 4.4: Heat maps of l_2-norms over the activation channels for Conv2D layers in the baseline ReLU network. *Top* row - input image is from class *boot*, *bottom* row - input image from class *bag*. One can see that those two objects that belong to different classes produce very similar heatmaps and as a result are both classified as a *bag* with confidence > 0.9.

Given such highly non-uniform distribution of feature importance, we want to measure how network performance is affected by "turning off" some random features. We implement this method simply with a dropout module:

$$x^{adv} = \texttt{dropout}_p(x) \tag{4.7}$$

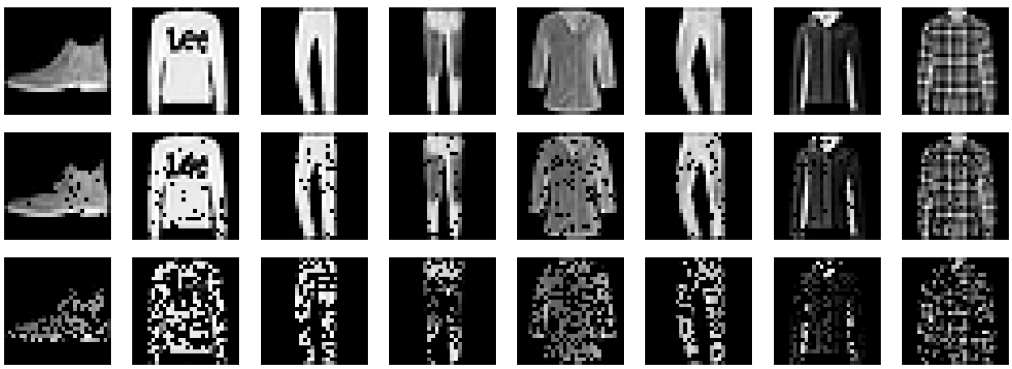

FIGURE 4.5: Adversarial examples generated with Missing Features approach: *top* row - original images from Fashion-MNIST dataset, *middle* row - generated images with $p = 0.1$, *bottom* row - generated images with $p = 0.5$.

Chapter 5

Experiments

5.1 Experimental setup

For our experiments, we consider the Fashion-MNIST dataset [42]. Since there is no published Deep Learning benchmark for Fashion-MNIST classification, we trained a baseline model with an all-convolutional architecture which is illustrated in Fig. 5.2. Achieved test accuracy is 90% (the best published benchmark - Support Vector Classifier - achieves mean test accuracy 89.7%). Results on baseline model robustness to adversarial attacks are showed and discussed in section 5.3.

FIGURE 5.1: Sample data from Fashion-MNIST [42]. The dataset consists of 28x28 grayscale images from 10 classes. The training set includes 60000 images, while the testing set - 10000 images.

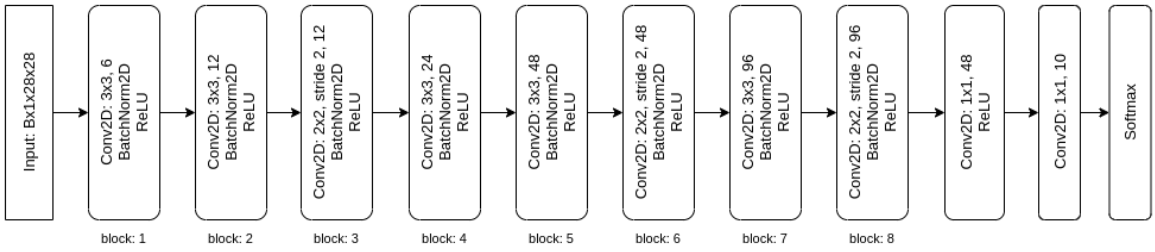

FIGURE 5.2: Architecture of baseline model. The same block architecture - Conv2D + BatchNorm2D + *activation* - is preserved in all experiments. However, in some experiments with stochastic neurons we *turn off* the batch normalization - *i.e.* eliminate BatchNorm2D from the model block and keep only Conv2D and activation function.

The main goal of experiments is to measure models robustness to adversarial attacks described in the chapter 4. We then explore further questions: does the introduced stochasticity impact model ability to generalize on the data and whether it can lead to other benefits. More explicitly, we look at the distributions of activations and gradients values during training of both deterministic and stochastic models as well as try to analyze features extracted by the models.

5.2 Implementation and Training Details

To conduct the experiments we used Python, more specifically Python3 programming language. All the models and stochastic modules were implemented using PyTorch library.
All the stochastic models were trained with Negative Log Likelihood:

$$l_{\text{NLL}}(\theta; x, y) = -\sum_{k=1}^{C} y_k \log \hat{y}_k(x; \theta) \tag{5.1}$$

where, x is the data sample, y is respective ground-truth label, θ is a vector of network parameters, $\hat{y}(\theta)$ represents predictions of model parameterized by θ and C is total number of classes. For ground-truth label y given as a one-hot encoding vector the NLL loss expression can be simplified as following:

$$l_{\text{NLL}}(\theta; x, y) = -\log \hat{y}_{i:y_i=1}(x; \theta) \tag{5.2}$$

From such definition one can see that optimizing NLL loss becomes equivalent to maximizing the likelihood of model predicting the correct label which in turn is equivalent to minimizing the empirical $0/1$ loss for a deterministic predictor [27].
The deterministic models were trained with Cross Entropy loss combined with softmax activation in the last layer [8].
Both the loss functions were optimized with the Adam optimizer [21]. Hyperparameters of the optimizer (*i.e.*, learning rate α and moving average parameters β_1 and β_2) are different for few experiments and are documented in respective experiment configuration files. We trained all the models for 100 epochs.
Code for experiments reproduction can be found here: `sol4ik/stochastic-predictors`

5.3 Results on Adversarial Robustness

For these experiments, we train deterministic models on a mixture of clean and adversarial examples generated with one-iteration FGSM, $\epsilon = 0.07$; a new set of adversarial examples is generated based on new model parameters after every 20^{th} training epoch. We also train stochastic models on clean data only. Later we test all the models against adversarial attacks discussed previously with different hyperparameters.

Looking at the plots of models performance on attacks (Fig. 5.3 - 5.5) one can see that stochastic models indeed perform relatively better that the deterministic baseline model, however they are not quite competitive with the network trained on adversarial examples. Besides this, we observe quite interesting behaviour of the models. Neural Network injected with Gaussian noise seems to be invariant to hyperparameters change for all the attacks. Stochastic Network with logistic noise also

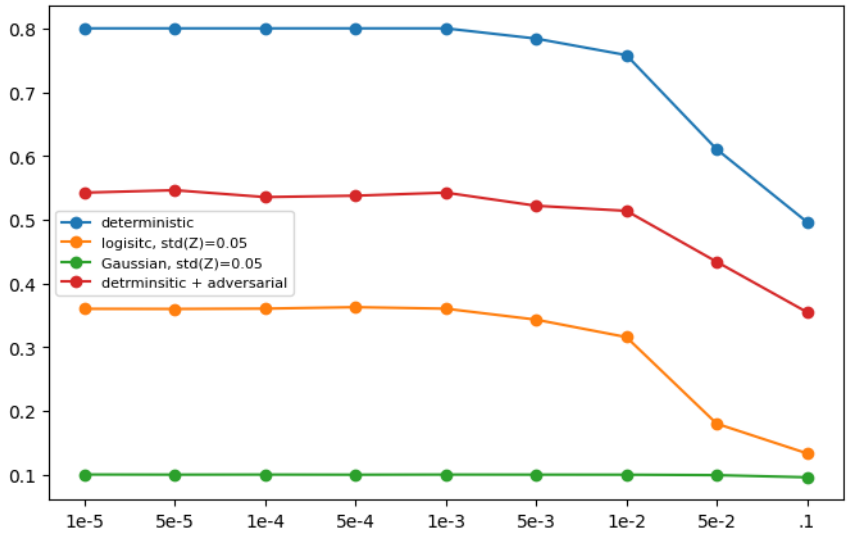

FIGURE 5.3: Test accuracy of models on targeted adversarial examples generated with FGSM. On x-axis - maximum perturbation ϵ, on y-axis accuracy. Stochastic models are injected with *additive* noise.

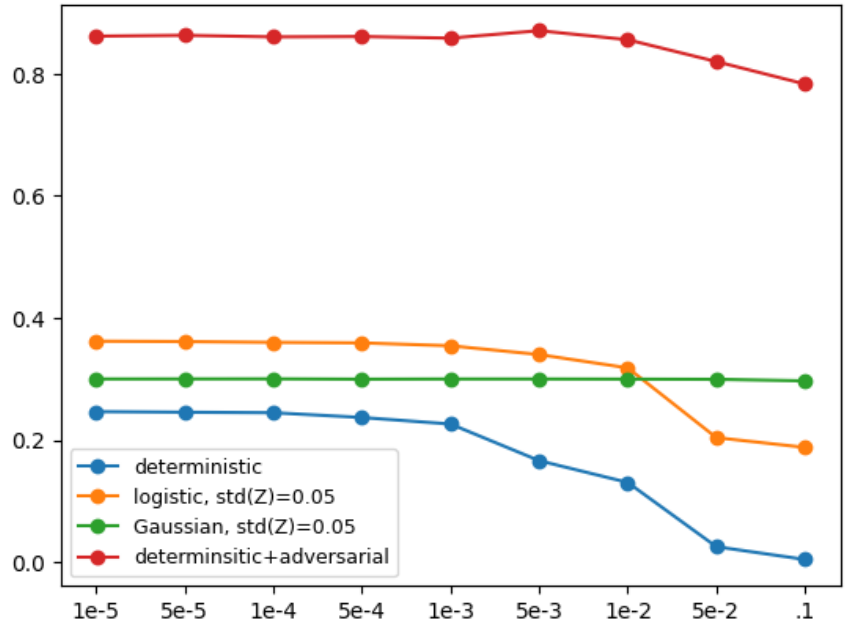

FIGURE 5.4: Test accuracy of models on untargeted adversarial examples. Stochastic models are injected with *additive* noise.

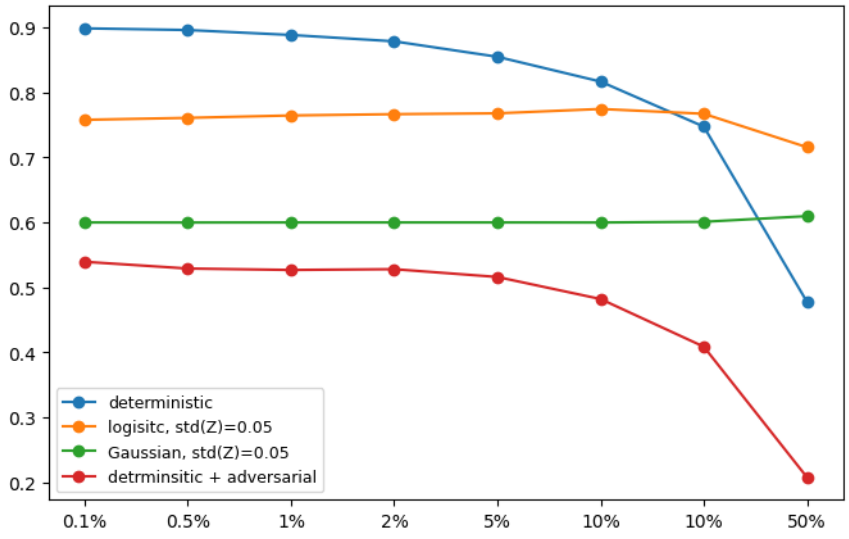

FIGURE 5.5: Test accuracy of models on adversarial examples generated with missing features attack. On x-axis - ratio of missing features p. Stochastic models are injected with *additive* noise.

shows almost constant accuracy except for last few attack runs which are the hardest. Model trained on mixture of clean and adversarial data performs well only on attack of same type which was used for adversarial examples generation, we even see a small increase in performance for attack with ϵ close to 0.07 which was used during training.
From these observation, we assume the following:

- The model trained on mixture of clean and adversarial images overfitted to the adversarial data, later in section 5.4 we discuss performance of this model on clean data and other results support this assumption.
- Stochastic Neural Networks injected with logistic noise show better performance compared to Networks injected with Gaussian noise. This assumption is also supported by the following experiments.
- Stochastic models learned data representation which is truly invariant to small perturbations in input examples, however this representation does not capture well true data distribution.

5.4 Impact of Stochasticity on Model Generalization

5.4.1 Model Accuracy

In section 3.2 we mentioned that improvement of on adversarial robustness of a DNN often comes at a cost of accuracy on clean test data. Even though the achieved adversarial robustness of stochastic models is not so far competitive with the results achieved with adversarial learning, one can see that stochastic models perform much better on the clean data compared to the baseline model trained on adversarial examples.

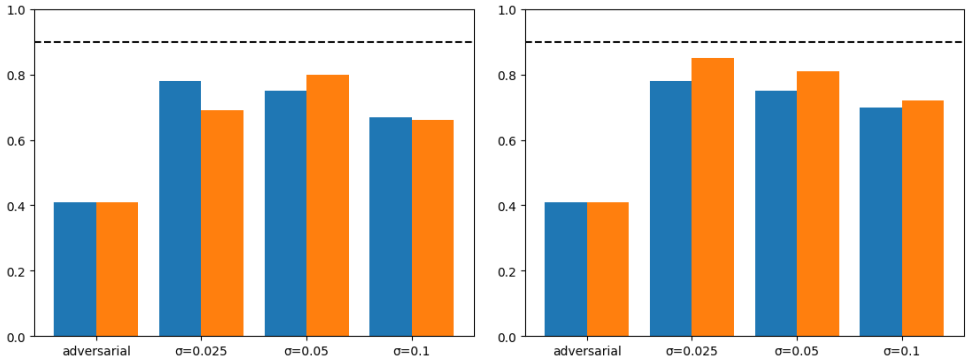

FIGURE 5.6: Test accuracy of Stochastic ReLU models. On *left* results of models injected with additive noise, on *right* - with multiplicative noise. For both plots, *first* bar column represents accuracy achieved by the baseline model trained with adversarial learning, *blue* bars represent accuracy of models with injected Gaussian noise, *orange* - with logistic noise, *dashed* line represents the test accuracy of a deterministic baseline model trained on clean data.

As one might expect injecting a model with noise of distribution with smaller value of standard deviation results in better performance on test data. Both ReLU and Binary models injected with logistic noise showed better accuracy. In case of Binary NNs, we explain this with the fact that Straight-Through Estimator for logistic noise becomes closer to the true gradient value; for Score Function Gradient Estimator-based neuron there is almost no difference in performance results.
It is worth noting that results discussed and illustrated in Fig. 5.6 and the rest of chapter 5 where obtained after a single run of model training. For better analysis one should conduct more training experiments and rather analyze mean value and standard deviation of accuracy values obtained.

5.4.2 Proneness to Overfitting

Due to high dimensionality of representation space and large amount of parameters Deep Neural Networks tend to overfit on training data. Formally, statistical model overfitting is defined as the production of an analysis that corresponds too closely or exactly to a particular set of data. In practice one might detect model overfitting by observing its performance on both training and test data: if model performance on train data is comparably better than its performance on test data, there is a high chance of overfitting.
For our specific case of Fashion-MNIST classification, we define model overfitting as training accuracy going over 90% while test accuracy stays incomparably low (*e.g.*, around 70%).
We observed no overfitting neither for deterministic models, nor their stochastic counterparts. However, as illustrated on Fig. 5.7, one can see that rate of accuracy growth on both train and test data for stochastic models is very small compared to the case of deterministic model which is a sign of more uniform learning.

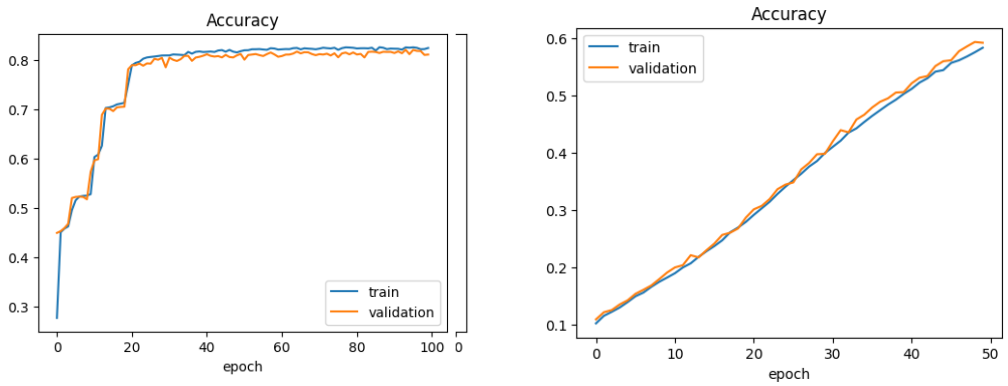

FIGURE 5.7: Learning curves for deterministic Binary model (*left*) and its respective stochastic counterpart (*right*); gradient estimator used for Stochastic Binary NN is STE.

5.4.3 Model Confidence on Correct and Wrong Predictions

Now we look at the confidence level of model predictions. Motivation for this experiment comes from the fact that even when making incorrect predictions Neural Networks tend to be very confident about them, *i.e.*, score returned by the network, which is later interpreted as a probability/confidence, is usually over 0.9. We again look into three different models: deterministic model trained in clean data, deterministic model trained with adversarial data and stochastic model, and check their predictions on clean test dataset.
For all models we observe almost identical (there are insignificant changes in the counts of confidence values lower than 0.5 for incorrect predictions) distribution of confidence level for both correct and wrong predictions. Histograms of respective distributions are illustrated in Fig. 5.8.

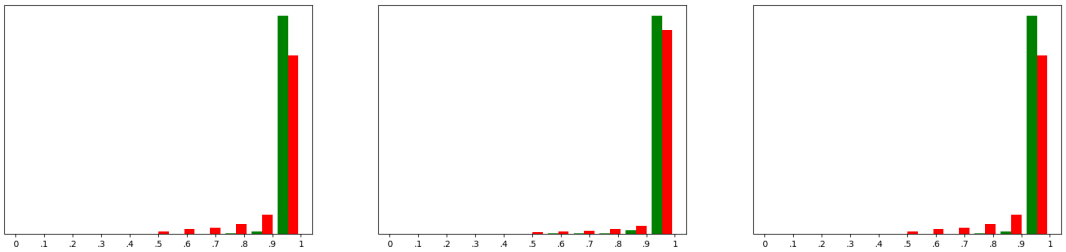

FIGURE 5.8: Histograms of model confidence. On *left* confidence distribution for predictions made by deteministic ReLU model, in *middle* - by deterministic ReLU model trained on mixture of clean and adversarial data, on *left* - by stochastic ReLU model with Gaussian noise ($\text{std}(Z) = 0.1$) injected.
For all histograms: in *green* - confidence on correct predictions, in *red* - confidence on wrong predictions.

5.4.4 Extracted Features Analysis

In our last experiments we try to compare and analyse features extracted by deterministic and stochastic models. We look at the features extracted by the models, more specifically we look at the outputs of the *block 8* (Fig. 5.2) and map them to $2D$ space with t-SNE mehtod for better visualization.

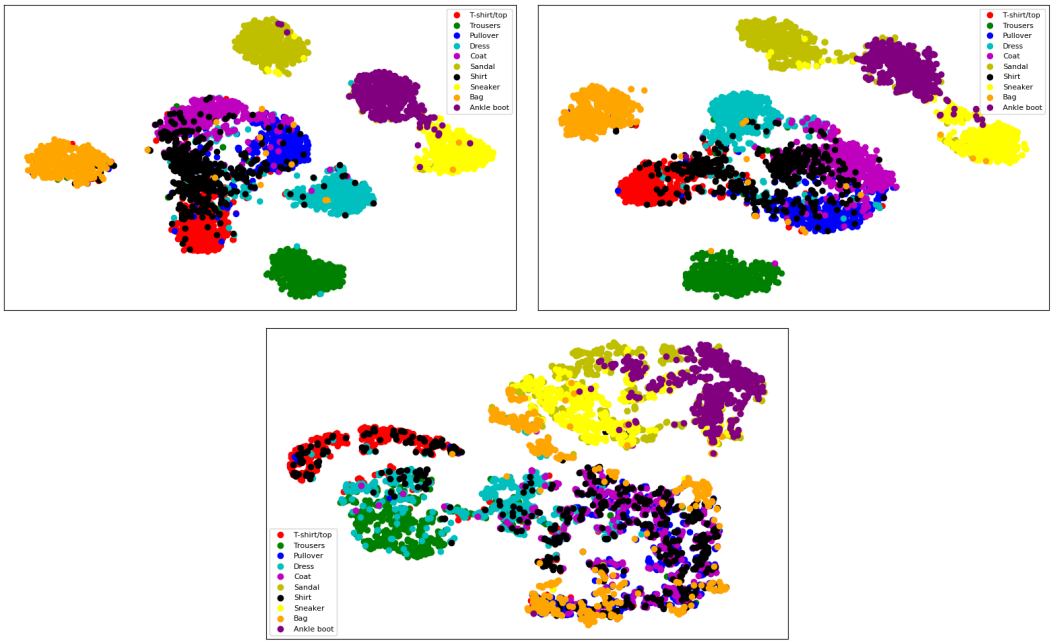

FIGURE 5.9: T-SNE plots of features extracted with deterministic (on *left*), adversarial (on *right*) and stochastic (on *bottom*) ReLU models. The stochastic model is injected with additive logistic noise, $\text{std}(Z) = 0.025$.

We observe quite interesting shapes of features clusters in data representation learned by stochastic networks. For both Gaussian and logistic noise clusters tend to have flowy shapes rather than solid ellipses as in case of determinsitic models (Fig. 5.9). Both additive and multiplicative noise act as a shaping factor and show to truly regularize representation space, however, as we can see thee is a need to try different noise distributions for possibly better results.

Chapter 6

Conclusions

6.1 Results discussion

In this work we have proposed and proved a hypothesis that stochastic relaxation of an Artificial Neural Network helps to increase its adversarial robustness. Experimentally we showed that injecting neuron pre-activations with additive or multiplicative noise from *logistic* distribution can help to increase model accuracy on adversarial examples generated with different methods almost $\times 1.7$ times while loosing only up to 10% of accuracy on clean data. We also compare performance of stochastic models with the model trained on adversarial examples which is current SOTA approach for building adversarial robustness. Even tough our method does not show competitive results compared to adversarial training, we show with other qualitative experiments that stochastic models show better generalizing abilities and potential for further improvement compared to deterministic models trained on either clean or adversarial data.
In sections 5.3 we show that data representation learned by Stochastic Neural Networks is indeed invariant to small data perturbations, however further fine-tuning and/or regularization is needed to learn representation which is closer to the true data distribution.

6.2 Future work

We briefly mention in section 5.4 that all results that were illustrated and discussed in this work were obtained after a single run of each training experiment. For better statistical analysis of noise injection impact on model performance it is needed to performs multiple runs of training experiments and analyze obtained sequence of values.
The noise distributions we chose for our experiments are too close value-wise and, thus, did not provide us with the desired variety of Stochastic Artificial Neuron models. We plan to explore different noise distributions (*e.g., uniform*) along with data preprocessing in order to get different statistics on clean dataset.
Our last proposal for further improvement is to inject each layer with noise of different distribution (*e.g.,* change std(Z) for each Conv2D layer according to change in pre-activations statistics) in order to perform better regularization of (pre-)activations in every layer.

Bibliography

[1] Lei Zhang et al. *Deep Learning for Sentimental Analysis: A Survey*. 2017. arXiv: 1801.07883.

[2] Milad Alizadeh et al. "A Systematic Study of Binary Neural Networks' Optimisation". In: *International Conference on Learning Representations*. 2019.

[3] Y. Bengio et al. "Greedy layer-wise training of deep networks". In: vol. 19. Jan. 2007.

[4] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *CoRR* abs/1308.3432 (2013). arXiv: 1308.3432.

[5] Joseph Bethge et al. *Back to Simplicity: How to Train Accurate BNNs from Scratch?* 2019.

[6] Nicholas Carlini and David Wagner. *Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods*. 2017.

[7] Paolo Crippa, Claudio Turchetti, and Massimiliano Pirani. "A Stochastic Model of Neural Computing". In: *Knowledge-Based Intelligent Information and Engineering Systems*. Ed. by Mircea Gh. Negoita, Robert J. Howlett, and Lakhmi C. Jain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 683–690.

[8] *Cross Entropy loss*. URL: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

[9] Zehao Dou, Stanley J. Osher, and Bao Wang. *Mathematical Analysis of Adversarial Attacks*. 2018. arXiv: 1811.06492 [cs.LG].

[10] Kevin Eykholt et al. *Robust Physical-World Attacks on Deep Learning Models*. 2018. arXiv: 1707.08945 [cs.CR].

[11] Lijie Fan et al. *When Does Contrastive Learning Preserve Adversarial Robustness from Pretraining to Finetuning?* 2021.

[12] Chris Finlay and Adam M. Oberman. "Scaleable input gradient regularization for adversarial robustness". In: *Machine Learning with Applications* 3 (2021), p. 100017. ISSN: 2666-8270. URL: https://www.sciencedirect.com/science/article/pii/S2666827020300177.

[13] Peter W. Glynn. "Likelihood Ratio Gradient Estimation for Stochastic Systems". In: *Commun. ACM* 33.10 (Oct. 1990), pp. 75–84. ISSN: 0001-0782.

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

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014.

[16] Douglas Heaven. "Why deep-learning AIs are so easy to fool". In: *Nature* 574 (2019). ISSN: 163-166. URL: https://doi.org/10.1038/d41586-019-03013-5.

[17] G.E. Hinton and R.R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science (New York, N.Y.)* 313 (Aug. 2006), pp. 504–7. DOI: 10.1126/science.1127647.

[18] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural computation* 18 (Aug. 2006), pp. 1527–54. DOI: 10.1162/neco.2006.18.7.1527.

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080.

[20] Sandy H. Huang et al. "Adversarial Attacks on Neural Network Policies". In: *CoRR* abs/1702.02284 (2017). arXiv: 1702.02284. URL: http://arxiv.org/abs/1702.02284.

[21] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014.

[22] Pushmeet Kohli et al. *Identifying and eliminating bugs in learned predictive models*. Mar. 2019. URL: https://www.deepmind.com/blog/identifying-and-eliminating-bugs-in-learned-predictive-models.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[24] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. *Adversarial examples in the physical world*. 2016.

[25] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[26] Kimin Lee et al. "Simplified Stochastic Feedforward Neural Networks". In: (Apr. 2017).

[27] Jiří Matas. *Lecture notes on Pattern recognition and Machine learning course*. Oct. 2021.

[28] Thomas Merkh and Guido Montúfar. "Stochastic Feedforward Neural Networks: Universal Approximation". In: *CoRR* abs/1910.09763 (2019). arXiv: 1910.09763. URL: http://arxiv.org/abs/1910.09763.

[29] Bruno A. Olshausen and David J. Field. "Sparse coding with an overcomplete basis set: A strategy employed by V1?" In: *Vision Research* 37.23 (1997), pp. 3311–3325. ISSN: 0042-6989.

[30] Tapani Raiko et al. "Techniques for Learning Binary Stochastic Feedforward Neural Networks". In: (June 2014).

[31] M.A. Ranzato et al. "Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition". In: July 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383157.

[32] Pradeep Rathore et al. "Untargeted, Targeted and Universal Adversarial Attacks and Defenses on Time Series". In: *CoRR* abs/2101.05639 (2021). arXiv: 2101.05639.

[33] Ruslan Salakhutdinov and Geoffrey Hinton. "Deep Boltzmann Machines". In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. by David van Dyk and Max Welling. Vol. 5. Proceedings of Machine Learning Research. PMLR, 16-18 Apr 2009, pp. 448–455.

[34] Hadi Salman et al. *Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers*. 2019.

[35] Ruhi Sarikaya, Geoffrey Hinton, and Anoop Deoras. "Application of Deep Belief Networks for Natural Language Understanding". In: *Audio, Speech, and Language Processing, IEEE/ACM Transactions on* 22 (Apr. 2014), pp. 778–784. DOI: 10.1109/TASLP.2014.2303296.

[36] Alexander Shekhovtsov, Viktor Yanush, and Boris Flach. *Path Sample-Analytic Gradient Estimators for Stochastic Binary Networks*. 2020. arXiv: 2006.03143.

[37] Pramila P. Shinde and Seema Shah. "A Review of Machine Learning and Deep Learning Applications". In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. 2018, pp. 1–6. DOI: 10.1109/ICCUBEA.2018.8697857.

[38] Christian Szegedy et al. *Intriguing properties of neural networks*. arXiv: 1312.6199v1.

[39] Florian Tramèr et al. *Ensemble Adversarial Training: Attacks and Defenses*. 2017.

[40] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256. ISSN: 0885-6125.

[41] Eric Wong and J. Zico Kolter. *Provable defenses against adversarial examples via the convex outer adversarial polytope*. 2017.

[42] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[43] Weilin Xu, David Evans, and Yanjun Qi. "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks". In: *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society, 2018. arXiv: 1704.01155.

[44] Penghang Yin et al. *Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets*. 2019. arXiv: 1903.05662.

[45] Piotr Żelasko et al. *Adversarial Attacks and Defenses for Speech Recognition Systems*. 2021. DOI: 10.48550/ARXIV.2103.17122.