BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

# IMPROVING ROBUSTNESS OF NEURAL NETWORKS AGAINST ADVERSARIAL EXAMPLES

ZLEPŠENÍ ROBUSTNOSTI NEURONOVÝCH SÍTÍ PROTI KONTRADIKTORNÍM VZORKÁM

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — MARTIN GAŇO
AUTOR PRÁCE

SUPERVISOR — RNDr. MILAN ČEŠKA, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2020

**Brno University of Technology**
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2019/2020

# Bachelor's Thesis Specification

22999

| | |
|---|---|
| Student: | **Gaňo Martin** |
| Programme: | Information Technology |
| Title: | **Improving Robustness of Neural Networks against Adversarial Examples** |
| Category: | Artificial Intelligence |

Assignment:

1. Study the problem of adversarial examples in the context of neural networks (NNs).
2. Study existing techniques for improving the robustness of NN against adversarial examples. Focus on methods (including formal methods) for effective generation of adversarial examples and retraining the NN using these examples.
3. Using existing frameworks for NN design, implement a retraining loop employing different methods for adversarial example generation.
4. Carry out a detailed experimental evaluation of the retraining process and assess how the process can improve the robustness of NN against adversarial examples. It is sufficient to focus on small and medium-size NNs.

Recommended literature:

- Zheng, S., Song, Y., Leung, T., & Goodfellow, I. 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 IEEE 2016.
- Weng, Lily, et al. "PROVEN: Verifying Robustness of Neural Networks with a Probabilistic Approach." *In Proceedings of the International Conference on Machine Learning*. pp. 6727-6736, 2019.
- Huang, Xiaowei, Marta Kwiatkowska, Sen Wang, and Min Wu. "Safety verification of deep neural networks." In *International Conference on Computer Aided Verification*, pp. 3-29. Springer, Cham, 2017.

Requirements for the first semester:

- Items 1 and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

| | |
|---|---|
| Supervisor: | **Češka Milan, RNDr., Ph.D.** |
| Head of Department: | Hanáček Petr, doc. Dr. Ing. |
| Beginning of work: | November 1, 2019 |
| Submission deadline: | May 28, 2020 |
| Approval date: | March 16, 2020 |

## Abstract

This work discusses adversarial attacks to image classifier neural network models. Our goal is to summarize and demonstrate adversarial methods to show that they pose a serious issue in machine learning. The important contribution of this work is the implementation of a tool for training a robust model against adversarial examples. Our approach is to minimize maximization the loss function of the target model. Related work and our own experiments leads us to use Projected gradient descent as a target attack, therefore, we train against data generated by Projected gradient descent. As a result using the framework, we can achieve accuracy more than 90% against sophisticated adversarial attacks.

## Abstrakt

Tato práce pojednává o kontradiktorních útocích na klasifikační modely neuronových sítí. Naším cílem je shrnout a demonstrovat kontradiktorní metody a ukázat, že představují vážný problém v strojovém učení. Důležitým přínosem této práce je implementace nástroje pro trénink robustního modelu na základě kontradiktorních příkladů. Náš přístup spočívá v minimalizaci maximalizace chybové funkce cílového modelu. Související práce a naše vlastní experimenty nás vedou k použití Projektovaného gradientního sestupu jako cílového útoku, proto trénujeme proti datům generovaným Projektovaným gradientním sestupem. Výsledkem použití nástroje je, že můžeme dosáhnout přesnosti více než 90% proti sofistikovaným nepřátelským útokům.

## Keywords

Neural networks, Optimization, Machine learning, Adversarial attack, Adversarial examples, Robustness, Adversarial machine learning

## Klíčová slova

Neuronové sítě, Optimalizace, Strojové učení, Kontradiktorní útok, Kontradiktorní vzorek, Robustnost, Kontradiktorní strojové učení

## Reference

GAŇO, Martin. *Improving robustness of neural networks against adversarial examples.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor RNDr. Milan Češka, Ph.D.

# Improving robustness of neural networks against adversarial examples

## Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of RNDr. Milan Češka, Ph.D. All the relevant information sources, which were used during the preparation of this thesis, are properly cited and included in the list of references.

. . . . . . . . . . . . . . . . . . . . . . .
Martin Gaňo
June 4, 2020

## Acknowledgements

I would like to thank my supervisor RNDr. Milan Češka, Ph.D. for his encouragement and continuous support throughout the work on this project.

# Contents

# Chapter 1

# Introduction

Neural networks plays important role in machine learning. Their success in solving some tasks almost completely replaced standard algorithms. Such tasks are for example image or speech recognition, malware detection, natural language processing, etc. The progress of neural networks and applying it in wide range of systems accelerated in past years and nowadays large amount of people in developed world are users of some neural network application.

Unfortunately, there is also a dark site of neural networks and may that issue may transform neural network applications into the weapon. Significant defect of neural networks is their vulnerability to malignant inputs, so called *adversarial examples*, since the algorithms of deep learning were not designed with defense mechanism. There is plenty other risks connected with deep learning including adversarial reprogramming, model stealing, data poisoning, leaking sensitive data(see Subsection 2.3.1), etc.

Professionals are for long time worried about using neural networks in safety critical applications. Safety critical systems managed by neural networks are employed in many areas including transport industries, medicine and defence [10], for example autonomous cars or operating robots. The reason of concern is the fact, that neural networks contain enormous number of trainable parameters and that is why they are too complex for human comprehension,. In other words, they are so called black-boxes for humans, therefore it is infeasible to explain or fix a misclassification, or another kind of error caused by seemingly faultless model.

This work focuses on one of the potential risks, specifically adversarial examples generated to fool neural network models for image classification. We explore methods for generating *adversarial examples*, so called *adversarial attacks* or *adversaries*. Adversarial example is an input almost indistinguishable from original sample, however it is misclassified [4].

Significant danger comes with computationally feasible methods for generating adversarial examples. Using such methods could the enemy affect the model prediction (breaking face detection, fooling autonomous car, etc.) and manipulate the target application. In some extreme cases this kind of attack could result in fatal consequences like serious endangerment lives or property of humans.

Adversarial attack is method that yields adversarial samples. There is plenty of adversarial attacks and for this work differs between *white-box* and *black-box* attacks. In other words attacks with and without direct access to parameters parameters of the target model. In practise, black-box attacks are usually more common simply because model are often a black-box for users. Many neural network applications user accesses only using API by passing input data and receiving prediction (or any other output of the model according to model type). Even we stated that black-box attacks are more common in this work we focus on white-box attacks. The general reason is that our main goal is not to propose best adversarial attack but to create and implement defence method. For designing defence and following evaluation are white-box adversarial examples much more suitable (see Chapter 4) because they are more sophisticated and could fool the model better than black-box samples. Another reason is that we must also consider situations when steals the model of some application (e.g. using reverse engineering or insufficient precautions of application) and may be able to attack using white-box method (which is usually more successful in fooling the model). Another reason for focusing on white-box attack is that every white-box attack could became a black-box attack when we use different target model, in other words when we generate adversarial examples against some model and use it against another one.

It would not be possible to resist all attacks used by enemies, because they are still improving their strategies and will probably never stop. However, we have no other choice, but to continue on this research, which leads to improving safety and robustness of all neural network models.

Task for this work is to design implementation of training robust neural network and evaluate its robustness against adversarial examples. For this purpose we summarized some adversarial attacks and we used them during training and evaluation of target model. All white-box attacks and all defences mentioned in this work are available to use in our framework. The one purely black-box attacks used in this work is AdvGAN. For final evaluation of robust model with this attack we use still same samples generated by AdvGAN.

## Contribution

The greatest contribution of this work is bringing tool that enables us to train model that is robust against adversarial examples. Also important advantage of our implementation is possibility to experiment with attacks and defences. The major improvement of existing solution is that our implementation could perform adversarial training for any kind of image classification neural network model. There is strong need for such tool and there is no available public implementation of defence methods for general usage. Notice, that our tool itself does not guarantee perfect results of training. After all, as any other ML task, every model is specific and require individual tuning and parameter selection. With wrongly selected training parameters is seldom achieved the desired result.

Notice, that the defence against adversarial attacks (or adversaries) differs according to type of attack. Related work and our experiments gave us hypothesis, that we are able to train model that is robust to all attacks (wrt. epsilon distance between adversarial example and natural sample), which are using only *first-order* information [11]. In other words, we expect to be able to create model which is resistant to whole class of attacks. We have experimentally proven the hypothesis with usage of some white-box attacks implemented by Ian Goodfellow et al.[1] and also with modification of Generative adversarial network designed for generating adversarial samples called AdvGAN [18]. All these attacks was not able to decrease accuracy of specially trained MNIST model under 90% without significant decrease of accuracy evaluated on natural data. It is important to state, that all attacks used for evaluating target model used only *first-order* information.

[1] https://github.com/tensorflow/cleverhans

## Related work

Szegedy at al. in 2014 [16] demonstrated some intriguing properties of neural networks, including but not limited to fact that one adversarial sample is often misclassified by more models. Neural networks are vulnerable to adversarial examples and interesting phenomenon is that some adversarial examples (for some dataset) are indistinguishable for human. This observations demonstrates what threat pose neural networks and gives us motivation to face the issue.

After that Ian Goodfellow et al. explained adversarial examples in [4]. They showed strategies that could generate adversarial samples to fool even the most precise neural networks with really little computation sources. Mentioned work proposed method like FGSM and Basic Iterative Method, that we adapted for our experiments and the most important defence method is based on these attacks.

Framework for safety verification of neural networks was proposed by Huang et al. [6]. The framework guarantees finding all existing adversarial examples by exhaustive search around the region of original sample implemented by technologies like Z3.

Research by Kurakin et al. [12] brought us ideas about adversarial training including his summary of attacks. They demonstrated transferability phenomenon and label leaking. Knowledge from their work was helpful to understand the difference between adversarial training using one-step attacks and iterative methods (with many steps). These observations will be explained later in this paper.

It is important to mention library *Cleverhans* which provides implementation of several attacks, however it currently misses any defence method. This is also the motivation to bring implementation that joins attack and defence methods. Madry et al. provided implementation of adversarial training for some concrete models for classification image samples from CIFAR10 and MNIST datasets [11] but this is also not sufficient for ML community and it is reason why we are presenting general implementation of adversarial training for any model in Python language using high-level library *Keras*.

For designing defence we adapted approach of adversarial training introduced by group of scientist from Madry et al. in their paper [11]. In this work, they proposed hypothesis, that if neural network is robust against PGD (Projected gradient descent) attack, it is also robust against every other *first-order* attack. This statement is experimentally proven in the mentioned paper and also in our work against number of adversarial attacks. The mentioned paper also studies relation between network architecture and capacity with robustness. They show, that the larger networks handle adversarial examples better than smaller networks with similar accuracy on natural data. The result of their work is a model that achieves more than 90% accuracy against state-of-the-art attacks (on MNIST data).

To demonstrate robustness of model against adversarial examples Madry et al. started the challenge of adversarial attacks [11]. Most successful black-box attack to MNIST robust model prepared by Madry et al. in this challenge was an extended implementation of Generative adversarial networks (GAN) [18]. GAN was firstly proposed by Ian Goodfellow et al. in 2014 [3]. Authors created framework called *AdvGAN* for perturbing original data to adversarial examples. *AdvGAN* framework is described later in this work.

## Structure of this work

Chapter 2 provides us an overview of the necessary theory for understanding motivation, mathematical expression of the problem and the solution. It contains background about adversarial examples as a security issue of neural networks, explanation and definition of robustness that is used for evaluation the defence methods, dividing attacks into categories and explanations of other important terms used in this work. In Chapter 3 we summarized all adversarial attacks used in this work either for evaluation models robustness or for training model against them. We also expressed every attack formally and explained advantages and disadvantages in terms of time consumption and strength of generated samples. Chapter 4 shows our consequent progress in creating defence against adversarial examples. It shows simple approaches like improving architecture or preprocessing input data for image classifier. In Chapter 4 we also explained how adversarial training works and formally expressed the minimax optimization problem which the adversarial training should use. In Chapter 5 we experimentally demonstrated hypothesis proposed in this work. The experiments supports motivation for robust model and finally evaluates robustness of the robust model trained using our framework. Chapter 6 concludes what we discovered and reached and on the contrary what are the issues about this work and also how should this research continue and what should the consequent work study.

# Chapter 2

# Preliminaries

In this chapter we explain and define important terms that will be used later in this work. Some of these terms are well-known in the field of machine learning and some are defined for purposes of this work. Before we define the research problem for this work we need to explain what are *adversarial examples* and *adversarial attacks* that generates them. For understanding practical motivation of this work we also summarized types of adversarial attacks. Experiments proving the risks of the attacks are in Chapter 5. For later evaluation of our results we are defining the term *robustness*. Full understanding of our solution requires some basic knowledge of neural networks training that is part of Chapter 4.

## 2.1 Adversarial example

Adversarial (*malignant*) example, is a data sample, which is almost indistinguishable from the original for humans eye, while the model misclassified it. Valid explanation is also that it is a little perturbed original data sample, that is misclassified by the target model. We need a formal and unified definition of adversarial example.

**Definition 1** *Let $x^{adv}$ be such n-dimensional vector that there exists such n-dimensional vector $x$ in set of original data, that $D(x^{adv}, x) < \epsilon$, while $t(x^{adv}) \neq C$, where $D$ is some metric, $\epsilon$ is a small constant, $t$ is target model and $C$ is correct class of $x$.*

**Example 1** *Consider target model $t$ and dataset of 2-dimensional vectors $X$, such that every element of $X$ is correctly classified by $t$. Let $x^{adv}$ be a 2-dimensional sample. There exists such sample in $x \in X$ that $L^{\infty}(x, x^{adv}) = 0.24$ and $x^{adv}$ is misclassified by $t$. Therefore $x^{adv}$ is adversarial example for $t$ with respect to metric $L^{\infty}$ and $\epsilon > 0.24$.*

According to Definition 1 not every adversarial example is an output of adversarial attacks, some of them are natural and would be even a part of training data. This work focuses mainly to samples that are generated by adversarial methods.

### 2.1.1 Metrics

In the context of this work we will use term *metric* with meaning of some distance function. The purpose of using metrics in this work is to find out distance between two samples, since there is a great need to retrieve information about the similarity between them. There is plenty of metrics and some of them (e.g. $L^2$) simulates similarity for humans eye very well.

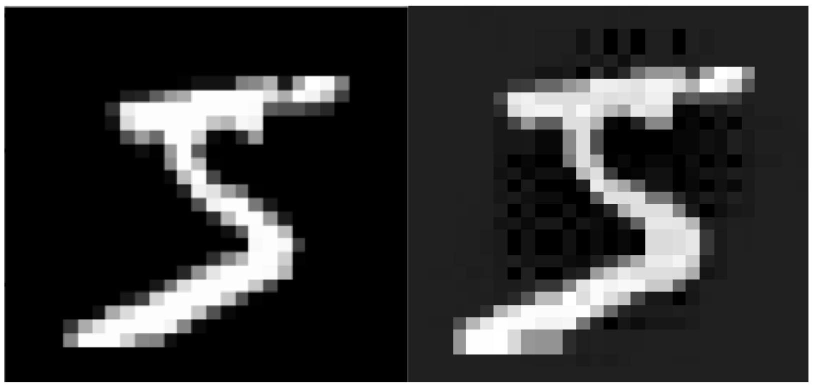

Figure 2.1: Visual comparison between natural and adversarial sample from MNIST dataset wrt. $\epsilon = 0.3$.

Typical metrics used in the previous work are $L^\infty$ and $L^2$ distances, since they properly simulates similarity [11].
$L^\infty$ metric, sometimes called *Chebyshev distance* between two vectors is defined as greatest value of their differences along any coordinate dimension.

$$d_{chebyshev}(x, y) = \max_i(abs(x_i, y_i)) \tag{2.1}$$

$L^2$ metrics is also called Euclidean distance.

$$d_{euclidean}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.2}$$

where $n$ is the dimension of the vector.

#### $L^p$-ball around point X

The term $L^p$-ball with midpoint X is frequently used term in this work. It means the n-dimensional space with shape of ball, while every point within this ball is meant to be similar to X wrt. norm $L^p$. As we stated before this work uses mainly $L^2$ and $L^\infty$ norms. A simple way how to imagine these $L^p$-balls is e.g. in 2 dimensions. In 2 dimensional space the $L^2$-ball around $X$ is a circle with midpoint in $X$ with radius value epsilon, thus we would consider every point in this circle as a similar point to $X$. Another example is the $L^\infty$-ball around $X$, this ball in 2 dimensional space would be the square with midpoint $X$. Since we will work in high-dimensional space it would not be possible to provide such simple visualization but we later show that the higher epsilon between sample is, the more different they are.

## 2.2 Robustness

Hiroaki Kitano provided definition of biological robustness. Robustness is a ubiquitous feature of biological systems. It ensures that specific functions of the system are maintained despite external and internal perturbations. [7] To generalize the biological explanation our understanding of robustness is the ability of system to handle errors that might affect it without changing the initial configuration.

There is need to adapt this description of robustness for the problem of robust optimization. Theoretically it would be really simple to define robustness using definition of adversarial examples this way: *The model is robust if it is not possible to generate any adversarial example.* Obviously this definition would be useless in most practical cases, because only models with 100% accuracy is robust according to mentioned definition. Our definition must cover this case but also must be much weaker to make it feasible to realize it, therefore we provide general definition in context of model optimization. *An optimal model is robust if it stays optimal under any allowed perturbation of input data.*

That statement is valid and useful for comprehension, however it does not provide us possibility to evaluate robustness formally since terms *optimal* and *allowed perturbation* are not well-defined. That is the reason why we also provide more specific definition that respects norm, epsilon and depends on required accuracy of the robust model. Notice, that in Definition 2 and also in whole this text we use the term accuracy (of the model). Every single occurrence of this term means the percentage of correctly classified samples in the related dataset. Because of that is the term accuracy always implicitly or explicitly connected with the dataset and the model.

**Definition 2** *Let $a_{nat}$ be an accuracy of optimal target model $t$ evaluated on natural data and $a_{adv}$ is an accuracy of $t$ evaluated on adversarial examples $X^{adv}$ generated by any method, while for each sample $X_i^{adv}$ of $X^{adv}$ exists $X_j^{nat}$ such that $d(X_i^{adv}, X_j^{nat}) < \epsilon$, where $d$ is some metric and $\epsilon$ is small constant distance. Then if there exists such $c$ that $a_{nat} - a_{adv} < c$, then model $t$ is c-robust wrt $\epsilon$.*

For using Definition 2 is important to choose optimal value of $c$ and consider outstanding $a_{adv}$ reached by successful model. Too small value of $c$ would cause inability to satisfy robustness conditions, on the other hand high value of $c$ would not satisfy practical requirements for robustness.

**Example 2** *Consider the target model $t$ with accuracy 98% evaluated on dataset $X$. Let $X^{adv}$ be the set of adversarial examples for $t$ wrt. $X$, metric $L^{\infty}$ and $\epsilon = 0.3$. Accuracy evaluated on $X^{adv}$ is 91%, therefore model $t$ is robust for $c > 7\%$.*

Notice that value for $c = 7\%$ in Example 2 refers to model with outstanding robustness against specific attack, since adversarial attacks could generate samples that decrease accuracy of successful model under 10% (see Chapter 5.2).

**Example 3** *Let the target model $t$ be a random classifier for 10 classes with accuracy $a_{nat} = 9.8\%$ evaluated on dataset $X$. Since random classifier could not be effectively attacked, let the accuracy evaluated on adversarial data $X^{adv}$ (with certain epsilon value) be $a_{adv} = 9.79$ thus random classifier is 0.01-robust wrt. some epsilon.*

As we can see in Examples 3, the random classifier as a target model is 0.01-robust what may seem as a brilliant result without knowledge of $a_{nat}$. We are using terms robustness and c-robustness through whole this text with similar meaning but beware the important difference. The term robustness is not well-defined and indicates model that is successful both for natural and adversarial data. On the other hand, term c-robustness always respects values of epsilon (measured using some norm) and $a_{nat}$, which need to be implicitly or explicitly known from the text.

## 2.3 Characteristics of adversarial attacks

In this section we described important characteristics of adversarial methods. This section is not explaining attack algorithms for every single attack but rather general properties and and risk connected to phenomenon of adversarial attacks. Detailed description of concrete attacks is located in Section 3.

Generally we consider finding adversarial examples as a *constrained optimization problem*. For the purposes of this work we are defining adversarial attacks as functions with variable amount of parameters and the function is outputting adversarial examples (see the subsection 2.1) respecting some constraints. The number of parameters the attack differs according to type of the attack and specific method. It is common for attacks to take as an argument input samples with their labels - this variable is essential for any kind of attack used in this work (however in some cases we could guess the labels or even generate adversarial samples from random noise). Sometimes the original data is used for training adversarial models (e.g. Adversarial generative networks) or in other cases the output data are perturbing directly from the original data (e.g. Fast gradient sign method). Another possible parameter is the target model. This is often used, but not necessary, because we could use our own target model and expect that the real target model will behave similar way (see Section 2.3.1). We distinguish two types of adversarial methods, according to information about the model that is passed as an argument. If the adversarial methods does not access the target model and use some different model for generating samples the method is called **black-box**. The second type is called **white-box** attacks. We consider attack as a white-box when an enemy accesses to all model parameters and whole model structure. Notice, that any white-box attack could became black-box - if we do not know the target model we could train and attack our own model and use the generated samples to attack the target (see Subsection 2.3.1).

Frequently used term in this work is also *order of the method*. Every usage of *n-th order* in this work will refer to *order* in **Optimization theory**. Generally, $n - th$ order methods is an algorithm for that $n - th$ derivation is the highest derivation it requires. For example, if the algorithm requires first and third derivation the method is third order.

### 2.3.1 White-box vs. black-box

As we stated above, we also differ between white-box and black-box attacks. Black-box attack does not use information about inner structure of the target model therefore it does not accesses weights and biases, neuron activation, architecture nor hyperparameters. They could only use different neural network model for its purposes. That is the reason why we often rely on so called *transferability phenomenon*, see subsection 2.3.1. In black-box methods training dataset could be sometimes known for the attacker, in these cases it use to be better to train on the same dataset to simulate target model better. If the attacker does not know the training dataset there is an alternative way how to find out training data called *Membership Inference Attack (MIA)* (see subsection 2.3.1). On the other hand, white-box attacks uses all the information about target model. There is also category called semi-white-box attacks but we will not discuss it in this work. In practise are black-box attack much more common because only small amount of models that manage some products or services are public. However, we need to discuss also white-box attacks, because they are more destructive and they could generate strong targeted perturbations (see subsection 2.3.2).

#### Membership Inference Attack (MIA)

Main purpose of MIA is to determine if some sample is in the models training data. To retrieve this information we often need confidences for each classes computed by the model [15]. MIA relies on the fact that model outputs higher confidence for samples from the training dataset.

This attack could be potentially serious security issue. Many public models are trained on sensitive data. If the attacker could be able to assign some personal record to the correct label indicating some illness or salary and it would lead to massive data leaks.

#### Transferability phenomenon

Many black-box methods are based on so called *transferability phenomenon.* According to [13] are adversarial examples transferable between neural network architectures i.e. black-box attacks often rely that generating malignant samples against target model will be also harmfull to another one.

According to Madry et al. [11] it is also useful for attacker to choose suitable model because training only against strongest models does not guarantee that the adversarial samples will be most transferable as they could.

### 2.3.2 Targeted vs. untargeted

Another partitioning of attacks is to categories *targeted* and *untargeted.* Targeted attacks are successful the target model classifies perturbed sample as some concrete class - the target. For example, the targeted attacks tries to perturb an old one or generate new sample that is an instance of class X but the target model classifies it as Y, where Y is the target. The goal of untargeted attack is just generate such samples that model classify with as low accuracy as possible, i.e. untargeted attacks are trying to maximize loss between input sample and perturbed sample so the perturbed sample is misclassified. In opposite targeted attacks are trying to minimize loss between perturbed sample and desired output class while satisfying the epsilon distance. [5].

### 2.3.3 Single vs. multi step

Before we show the difference between single and multi-step methods we explain what does it mean an attacking *step.* A *step*, or iteration, in the context of adversarial methods, means a single update of the sample. As we can see in the Figure 3.2, it is absolutely not correct to state that the further from the original sample perturbation is, the higher the loss is. The step is typically performed by computing the optimal direction and moving by $\gamma$ in that direction. E.g. FGSM tries to perform the largest possible step, therefore for FGSM $\epsilon = \gamma$.

Single step attacks are right as their name implies performing just one step of perturbing an input sample. Multi step attacks are performing various number of steps according to specific method. Generally, these multi step methods are trying to find local loss maximum, while not moving outside the $\epsilon$-ball around the original sample. This can include few times restarting the method from random starting points and compare results after method converges to local loss maximum. As we experimentally proved in 5.2 all the local maximums found by PGD method are almost the same. Outputs of multi step methods are way more sophisticated and harmfull. Single step methods are significantly faster than multi step methods, while still destructive enough (see 5.2). When we training mode against adversarial examples it is useless to train against single step samples, because in that case model use to overfit to the specific method. Madry et al. [11] stated that this could be caused by simplicity of perturbation which can model easily learn and overfit.

Notice, that all attacks could be divided into single and multi-step. In context of this work we could say that FGSM is the step and other methods are using iterations of FGSM. On the other hand, AdvGAN is neither single nor multi-step attack because it is trained model that generates samples.

# Chapter 3

# Adversarial methods

Important step for creating robust model is to study and understand adversarial methods used for attacks. In this chapter we summarized and explained all adversarial attacks used in this work. Every attack mentioned in this chapter is also provided in our implementation.

Method we are using takes as an input original sample and outputs perturbed input sample, while the perturbation is computed with the goal not to be classified the same as an original sample, respectively to be classified as a desired output in the context of targeted attacks. It is also possible to generate adversarial samples from random noise using trained generator or adding some noise or object to sample for fooling target model.

## 3.1 Fast gradient sign method (FGSM)

The method was proposed first time by Ian Goodfellow et al. in 2015. It is based on computing an adversarial image by adding a perturbation with magnitude one pixel in the direction of the gradient, thus this method is certainly white-box attack. Formula for computing result sample is

$$X^{adv} = X + \epsilon * sign(\nabla_x L(X, Y)) \tag{3.1}$$

where $X^{adv}$ is adversarial sample, $X$ is original sample, $\epsilon$ is size of perturbation, $L$ is loss function, $Y$ is label of original data and $sign(x)$ is function that returns $-1$ if $x < 0$; 0 if $x = 0$ and 1 if $x > 0$.

The method is very efficient in terms of computational time, because it is computed just with one single step. [12]

## 3.2 Projected gradient descent (PGD)

Another important term not only for this work but also for whole deep learning field is *gradient descent*. It is an *iterative, first-order, optimization* algorithm for finding local minimum of a differentiable function. Almost all training and many adversarial methods in deep learning are based on gradient descent. Purpose is to find local maximums of models loss function for optimizing its performance. Since generating adversarial examples given maximal epsilon distance from an original sample, it is a constrained optimization problem. For this purposes we use modification of gradient descent called projected gradient descent.

Projected gradient descent (PGD) is the most successful white-box method mentioned in this work. Cause this is a key method for improving robustness of the model, it will be

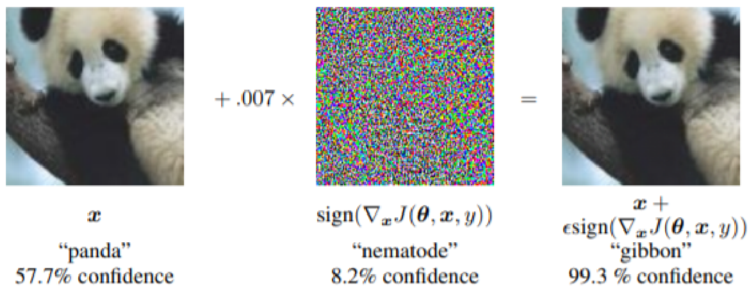
+ .007 ×
=
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 3.1: How adversarial attack modify image. Adapted from [16]

described more detailed and with special attention. Similarly as FGSM, we are using PGD to find such adversarial inputs, that maximizes the loss of the model, while the distance between generated data and original data according to certain metric does not exceed $\epsilon$. Correctly chosen $\epsilon$ (with correctly chosen norm) guarantees that generated examples will be similar, or even indistinguishable from the original samples.

The algorithm is a straightforward extension of FGSM 3.1 that is applied in iterations with step size $\gamma$ and the algorithm clips pixel values after each step - this ensures keeping generated sample within $\epsilon$-ball of the original data sample [9]. Let $X_n^{adv}$ be an adversarial example computed with n iterations. We first set $X_0^{adv}$ to some point within the $L^p$ ball (in our case $p = \infty$) with radius of $\epsilon$. Then we make a gradient step with size $\gamma$ in the direction of greatest loss and project back to $L^p$ ball [1] as in equation 3.2 if needed. This projection back performs function *clip*. Then we apply equation 3.2 until the stopping criteria is satisfied. The stopping criteria in our implementation is number of iteration of PGD algorithm but it may also be the value of loss differences between steps or time limit. To compute $X_n^{adv}$ we use this formula:

$$X_n^{adv} = clip_{X,\epsilon}\{X_{n-1}^{adv} + \gamma * sign(\nabla_x L(X_{n-1}^{adv}, Y))\} \tag{3.2}$$

where $sign(x)$ is function with same meaning as in Equation 3.1 and $clip_{X,\epsilon}(x^{pert})$ is a function that returns $x^{pert}$ if $D(X, x^{pert}) < \epsilon$, otherwise it returns nearest point to $x^{pert}$ that is within $\epsilon$-ball around $X$.

This method also requires access to model parameters, so we consider it as a white-box attack. For better understanding the method see the figure 3.2.

### Basic Iterative Method (BIM)

This method is a special case of PGD. For perform BIM we do not set starting point randomly. In this work will be starting point of BIM always set to original sample.

$$X_0^{adv} = X \tag{3.3}$$

Another steps of algorithm are same as for PGD.

[1]projecting back to $L^p$ ball means to move to the closest point inside the $L^p$ ball

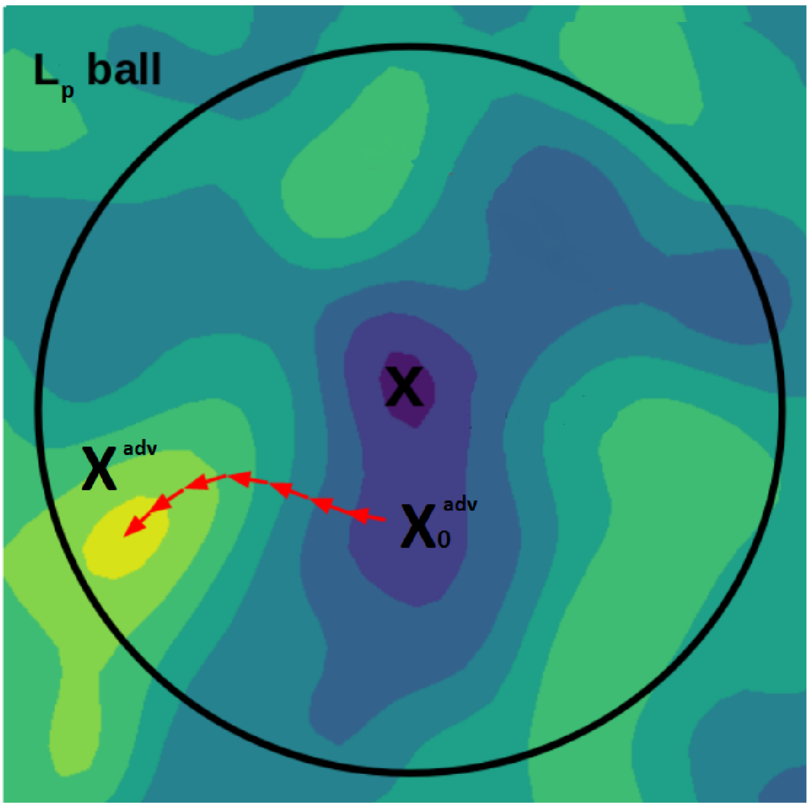

Figure 3.2: Visualization of loss function around original sample X. PGD after $n$ iterations found the local loss maximum. As you can see the starting point of PGD is random point in the $\epsilon$-ball around the original sample. The method reached the highest loss (yellow area). Since the Figure contains more local maximums there is a chance that starting point from another point would lead PGD to smaller loss value. Figure is adopted from [8].

## 3.3 Momentum iterative method (MIM)

The last white-box method explained and used in this work is MIM. MIM is the extension of PGD solving the same constrained optimization problem but the gradient descant used in the method is enriched with momentum. [2].

Generally, the gradient descent with momentum is used for acceleration the descent [14]. It is main principle is to accumulate velocity vector in the direction of gradient across the iterations. Steps of the method are not in the direction of steepest descent, but we also consider the previous directions. The knowledge of accumulated gradients prevents us to reach poor local maximum and helps us to flatten the curve of the gradient descent path thus it make descent faster.

For comprehension imagine a ball rolling down the hill. The ball is not rolling in the direction of the steepest descent but its direction is affected also by its speed. If it rolls fast enough it could even roll up the hill for some time. This could be useful if the ball get into some poor local minimum but it still has enough energy to continue. Finally, the ball stops in some valley (local minimum).

Similarly as PGD and BIM this is also a multi-step algorithm. Starting position for $X_0^{adv}$ is the original sample like in BIM, which is a special case of PGD (see Equation 3.4). Before the iteration starts we need to initialize velocity accumulator $g_0$:

$$g_0 = 0 \tag{3.4}$$

The difference between MIM and PGD is already mention accumulation of velocity vector. Before computing the perturbation in every iteration we accumulate the velocity by updating $g_n$:

$$g_n = \lambda * g_{n-1} + \nabla_x L(X_{n-1}^{adv}, Y) \tag{3.5}$$

where $\lambda$ is the number between 0 and 1 including and it represents decay factor for accumulation. Other symbols are used with same meaning as in Equation 3.2.

For computing $X_0^{adv}$ we apply this formula:

$$X_n^{adv} = clip_{X,\epsilon}\{X_{n-1}^{adv} + \gamma * sign(g_n)\} \tag{3.6}$$

where $g_n$ is accumulated velocity vector and other symbols are used with same meaning as in Equation 3.2. The stopping condition for the algorithm is same as for PGD number of iterations.

Notice that BIM is a special case of MIM with $\lambda = 0$, since the starting position is the original sample and with $\lambda = 0$ gradient is not accumulated.

## 3.4 GAN for generating adversarial examples

In 2019 was proposed framework AdvGAN in [18] and took the first place in MNIST Challenge[2] in black-box category. This is the only one *black-box* attack described and used in this paper. The purpose of the this framework is to train *generator* to be able to generate perturbations from the original data - after the model is trained, there is no need to access target model. This is also a reason why is *AdvGAN* much more effective than other adversarial attacks. When the generator is trained, we can generate perturbation for any sample using feed-forward network, while other methods must apply gradient descent for every single sample. Implementation of AdvGAN is an important part of our framework that enables us experiment also with this black-box attack.

### Architecture

Unlike many other GAN-based architectures, our generator does not take as an input random noise, but the original instance. Output of $G$ is a noise that is added to original sample (see Figure 3.3). The perturbed sample $X + G(x)$ goes into $D$ along with original sample and $D$ distinguishes if $x + G(x)$ and $x$ are similar. At the same time $x + G(x)$ takes target model $f$ as an input and returns the loss function $L_{adv}$, which represents the distance between predicted sample and target class. The output of $D$ is the adversarial loss [3]:

$$\begin{aligned} L_{GAN} = \\ &\mathbb{E}_{X\sim p_{data}} log\ D(x) \\ &+\mathbb{E}_{X\sim p_{data}} log(1 - D(x + G(x))) \end{aligned} \tag{3.7}$$

The loss of target model is:

$$L_{adv} = \mathbb{E}_{X\sim p_{data}} l_f(x + G(x), t) \tag{3.8}$$

[2]In the time of writing this work it is still the best submission

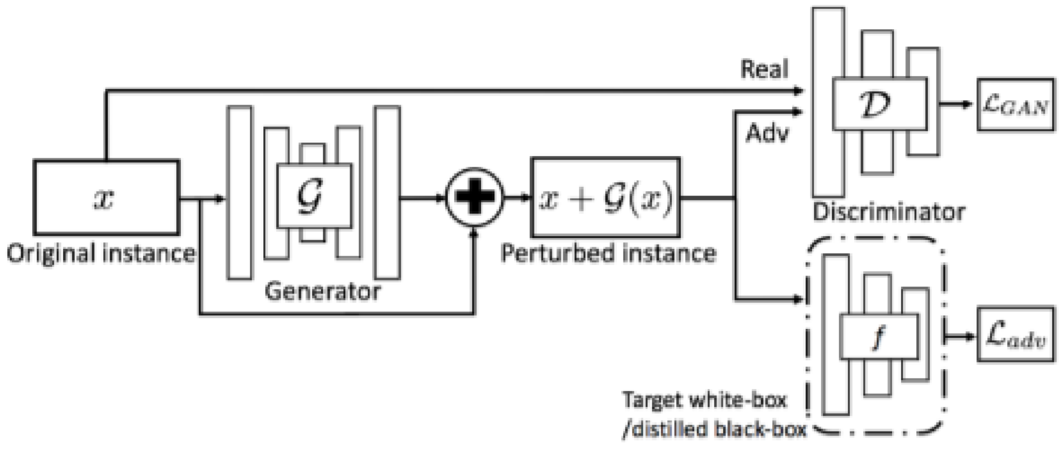

Figure 3.3: Architecture of AdvGAN. Figure is adopted from [18]

where $t$ is target class, $f$ is target model and $l_f$ is loss function of target model. This loss is used for targeted attack, untargeted attack could be performed by maximizing distance between prediction and label of the sample. We also need to constrain the noise generated by $G$ to prevent exceeding previously defined $c$ (see Definition 1). For this purpose we use hinge loss, what is usual practise in previous work [1] (unlike referenced work we use $L^\infty$ norm instead of $L^2$). Hinge loss is formally expressed as follows:

$$L_{hinge} = \mathbb{E}_{X \sim p_{data}} max(L^\infty(G(X)) - c, 0) \tag{3.9}$$

Finally, the generator is trained on objective function

$$L = L_{adv} + \alpha * L_{GAN} + \beta * L_{hinge} \tag{3.10}$$

where $\alpha$ and $\beta$ are experimentally chosen constants and represents relative importance of loss functions.

# Chapter 4

# Defence

The important contribution of this work is implementation and evaluation of defence methods against attacks. This chapter summarizes methods used for increasing accuracy of neural networks against adversarial examples without radical decrease of the accuracy evaluated on natural data.

Our first approach for research of defences was that adversarial examples are results of overfitting to natural samples. This approach could be formally express as a problem of finding parameters $\theta$ for minimizing the loss function $L$:

$$\underset{\theta}{argmin}\ L(x, y, \theta) \tag{4.1}$$

where x and y are vectors of samples and their labels. There are no doubts that this problem is solved by standard neural network training, however, for some reason it does not work for specially perturbed - adversarial samples.

We noticed that neural network models with even only 3 layers could achieve almost 100% accuracy for MNIST dataset. This observation shows us that MNIST dataset is very simple and we need to beware this knowledge through whole this work. After performing some attack we find out that shallow neural network models with low capacity are more vulnerable to adversarial examples comparing to deeper and larger models. As a result we discovered that capacity increases c-robustness and it is also approved by research of Madry et al. [11].

Unfortunately, only increasing capacity does not improve the robustness of model enough for practical requirements. Architectures with optimal capacity are still only slightly more accurate than random model with uniform distribution for MNIST dataset and their vulnerability to attacks is high. On the other hand, in this point of research we expect that creating robust model will require sufficiently large capacity, because desired model need to learn larger amount of information.

Another view on the defence against attacks is removing the noise added to samples. A suitable method for reducing noise in image samples is using filters. This method is easy to apply and consumes low amount of commutable resources. Later in this chapter we described that filtering could achieve impressive results for MNIST dataset but not as good for CIFAR10.

Previous experience leaded us to approach of creating model that is trained against wide range of data to achieve robustness. This would certainly require model with large capacity and extended data source. The method we used for extending our data is *data augmentation.* This approach helped us train better model against both natural and adversarial data but the improvement was again not sufficient. There were still many high local error maximums in models loss function that were found by attacks.

The last approach we decided to use was the computationally hard process of training model against adversarial examples. This includes training and performing adversarial attacks which makes it hard in terms of resources. This approach is based on minimizing maximization the error, i.e. trying to find adversarial examples before the attacker and retrain the model against it. Formally it could be expressed as the extension of standard training where we aim not just minimize the loss function but we minimize the loss function wrt. local loss maximums:

$$\underset{\theta}{argmin}\ E_{(x,y)}[\underset{\delta}{max}\ L(x+\delta, y, \theta)] \tag{4.2}$$

where $E$ is the maximized error of loss function $L$ wrt. to perturbation $\delta$ which is constrained by $\epsilon$, $\theta$ are parameters of the model, $x$ and $y$ are samples with related labels. The maximization of the error is performed by some adversarial attack.

In other words, this defence strategy aims to learn the malignant perturbations generated by a specific attack. At the beginning we expect to improve c-robustness at least against some specific attack and since the vast majority of state-of-the-art attacks are based on first-order gradient descent we expect all malignant perturbations will be similar and the model will resist them if will be trained correctly.

To avoid computationally hard process of repetitive applying multi-step attack, we decided to train our model against FGSM. This process yields relatively accurate model against FGSM samples but not against other methods. Perturbations generated by FGSM are probably too simple and the model overfits to them. Another possibility was training against PGD samples. This methods starts from random points and yields relatively complex perturbations so we expected it will prevent overfitting.

Training MNIST model against PGD samples leaded to creating highly precise model with accuracy against both natural and adversarial data high above 90% evaluated against few adversarial attacks. The training was relatively fast and did not require massive amount of training iterations. The result model achieved similar results as thresholding in terms of accuracy and after training model we do not need to filter the image since alone forward propagation yields correct results.

There is also strong need to prove that the method works for larger and more complex samples as CIFAR10. We evaluated the method on CIFAR10 and find out that it is also very successful, however, with weaker c-robustness than in case of MNIST dataset. Another interesting observation was the required resources for training robust model against CIFAR10. The training process lasted long time and required large amount of iterations, while every iteration lasted few-times longer time than for MNIST because of larger images. Finally, we decided that adversarial training with reference attack of PGD is satisfying our requirements for models robustness.

## 4.1 Filters

Since in this work we are discussing mainly image samples, this chapter will describe filters in the context of computer graphics. This defence method is certainly sufficient for use only for some kinds of adversarial examples as image or sound samples. On the contrary, filters would not be useful for text or malware adversarial examples.

Filtering is a method that modifies size, colors, shading and other properties of the sample. Purpose of filtering could be addition of some effects (e.g. smoothing). Since many adversarial methods add some noise to the sample, removing that noise could lead the model to classify sample correctly.

### 4.1.1 Thresholding

Thresholding is simple filter often used for image segmentation and color reduction. As we experimentally demonstrated in Section 5.3, thresholding is efficient defence against adversarial methods for datasets with low complexity (e.g. MNIST). Our approach of thresholding implementation was to set value of each color channel to minimum if its current value was under the threshold, otherwise to maximum.

Let $p_{i,j}$ be the pixel on the position $[i, j]$ and $p^{new}_{i,j}$ be the pixel in the result picture on the same position. Then we compute value of the $p^{new}_{i,j}$ as follows:

$$\mathrm{p}^{new}_{i,j} = \begin{cases} 0, & \text{if } \mathrm{p_{i,j}} \leq 0.5 \\ 1, & \text{if } \mathrm{p_{i,j}} > 0.5 \end{cases} \tag{4.3}$$

### 4.1.2 Median filter

Median filter is non-linear filter commonly used for removing noise. The principle is to iterate through the image matrix and replace values with median of their neighborhood. The size of the neighborhood depends on filter size. Filter size is normally a value that determines size of filter in all dimensions of the image.

Formally, let us consider filtered object as a two dimensional image. Let $p_{i,j}$ be the pixel on the position $[i, j]$ and $p^{new}_{i,j}$ be the pixel of the result image on the same position. The $N^f_{i,j}$ is a vector containing pixels in the $f$-neighborhood around $p_{i,j}$ where $f$ is a size of the filter. $f$-neighborhood around $p_{i,j}$ Then we need to compute $p^{new}_{i,j}$ for every $p_{i,j}$ as follows:

$$p^{new}_{i,j} = med(N^f_{i,j}) \tag{4.4}$$

## 4.2 Adversarial training

As we showed in Equation 4.2, the problem is defined as minimizing the maximized loss function. This method includes iterative attacking and learning samples generated against current model by applying certain attack, so called *reference attack*. Our aim is to flatten the loss function and reduce all local loss maximums for all samples in epsilon-ball around training samples. There will be no guarantee that some high loss maximum, or high value does not exists - it just need to be hard to find. To avoid computationally hard process of generating adversarial examples we unsuccessfully used data augmentation to extend dataset and train on wider range of samples. This section describes training against augmented data and also adversarial training against FGSM and then PGD as reference attacks.

### 4.2.1 Data augmentation

Data augmentation is a method for extending dataset and increasing dataset diversity by adding modified samples. There is plenty of possibilities how to modify samples, while they are not very different to original class. Common method for data augmentation is using various image transformations like changing scale, rotation, perspective, cropping, padding, horizontal flipping, etc.

This method is promising since it yields data samples in some unspecified epsilon distance around the original sample and we hoped it will retrain local loss maximums near to original samples. Process of learning augmented data is commonly used for improving accuracy of neural networks and consist of extending dataset by generating augmented data and then standard learning the dataset.

In subsection 5.4.1 we showed using MNIST and CIFAR10 datasets that data augmentation is also slightly improving c-robustness but actual results are still not satisfying the research requirements and are even worse than random classifier with uniform distribution. According our experiments improvements achieved by data augmentation could be achieved also by changing number of training epochs.

### 4.2.2 Training against reference attack

Since data augmentation failed to improve c-robustness of the model, we need to train model against some predefined adversarial attack (reference attack). The aim of adversarial training is to reduce high loss function values around original samples from training dataset by learning problematic samples. Process of training is similar to standard neural network training but there are some necessary differences.

The standard neural network training uses some predefined training dataset directly. On the other hand, adversarial training uses perturbed training data for learning model. In every training iteration the reference white-box attack generates batch of perturbed samples from training set against the target model. The model uses the batch of perturbed data for parameter optimization and the process continues. That means that samples for learning are probably similar but only rarely the same, on the contrary standard neural network training repeats samples in every epoch.

For better understanding the process of adversarial training see the following high-level python code of function for adversarial training.

```python
def adversarial_training(target_model, reference_attack, dataset, iterations):
  for _ in range(iterations):
    natural = get_next_batch(dataset)
    perturbed = generate_perturbations(target_model, reference_attack, natural.x)
    target_model.train_on_batch(perturbed, natural.y)
```

For clarification, parameter target model is the model for what we are improving c-robustness. Reference attack is some white box attack that generates adversarial samples for models relearning. Dataset contains natural training data for the model. Parameter iterations means the amount of iterations for training model on one batch therefore one training iteration means one optimization of model parameters (batch training).

# Chapter 5

# Experiments

This chapter contains experiments performed in order to tune our framework and to demonstrate its functionality. Experiments are also reasoning the usage of values of parameters and metrics. This work focuses on adversarial examples and defences against them generated by untargeted attacks which main purpose is to fool the model as much as possible. Our generated samples are perturbed original samples, while the size of perturbation is constrained by maximal epsilon. The target models used in this work are neural network models for image classification.

Since that we are focusing on public image datasets *MNIST* and *CIFAR10*, however, training robust network is a complex task and every neural networks problem needs unique parameters. Result robustness also relates to the architecture of a network, i.e. some architectures are not able to be robust (e.g. because of small capacity). *MNIST* and *CIFAR10* are simple image datasets. *MNIST* dataset used in this work contains 70 000 samples, including 60 000 samples used for training and 10 000 samples used for testing our solution and accuracy of the model. Shape of samples from *MNIST* is $[28, 28]$ with one color channel. *CIFAR10* dataset used in this work contains 50 000 and 10 000 testing samples. *CIFAR10* dimensions are $[32, 32]$ with 3 color channels.

Experiments in this chapter answers important research questions and explores approaches for solving problems defined in the work. The base problem is creating a model that is relatively resistant against adversarial attacks. For completing these tasks we first need to validate few assumptions and perform observations leading us to robust model.

First of all we need to demonstrate adversarial methods to show motivation for creating robust model. We assume that adversarial attacks could generate samples that clearly belongs to certain class, however neural network model classifies them with high probability wrongly. In other words, we expect that naturally trained model would achieve small level of c-robustness (with high c value).

For this purposes we need to show that the state-of-the-art network without special training would fail against attacks. This work shows various attacking methods and comparison of their time consumption and success against several models. This comparison would show us the best attack for specific model or dependency between time and model architecture wrt. attack.

For performing attack we need to set few parameters of the specific method. There is always one parameter shared by all the methods and it is the epsilon wrt. norm. Epsilon represents maximal distance between the generated samples and the original samples wrt. some norm and the distance should simulate similarity.

Since this work considers image datasets we need to verify that adversarial attacks are able to generate samples similar to original data. As the level of similarity is determined by value of epsilon wrt. norm it is also important to choose suitable norm and value for epsilon. Important property of epsilon wrt. norm is that every sample within epsilon-ball around the original image is at least strongly similar (or even indistinguishable) for human eye.

One of key methods used in this work is PGD. This method is not deterministic because uses random starting position. Therefore it is possible that some local maximums found by PGD could be significantly lower than local maximum found by restart of the method. It is also required to check this hypothesis and act accordingly in following work.

After illustrating that adversarial attacks are harmful for the model we proposed solutions. Since current state is that models accuracy against adversarial examples is often worse than a random model and our goal is to significantly improve the c-robustness. Firstly we considered the adversarial examples problem as an overfitting to natural data and we tried use the architecture for better generalization. This approach was helpful in terms of attacking time consumption and c-robustness, however attack was still feasible to perform and caused still significant decrease of accuracy so the method alone is insufficient. We experimentally showed that

Another proposed defence method was filtering. As we can see in the Figure 5.1, adversarial attacks are adding noise to the image and the noise is fooling target model. Our idea was to reduce the noise using image filters. We used two simple filters that reached certain level of success against attacks, however this solution was non transferable between models and specific filters worked only for specific dataset. This method is definitely not a final solution to our problem but it worths to study and in some datasets it could resolve adversarial examples problem.

Our final approach was to prepare the model against adversarial examples by special - adversarial training. This training should minimize the loss against adversarial examples and our hypothesis was that it will keep high accuracy against natural data after training on adversarial samples. For applying this approach we needed to determine plenty of variables like type of reference attack (or attacks) for generating adversarial examples, epsilon, step size, number of attack step, etc.

Firstly we were trying to avoid performing difficult process in terms of time so we decided to extend out dataset with augmented data and trained model against them. Result was again slightly better robustness without changing the attacking time, however the improvement was still not satisfying our requirements.

As the FGSM is the least time consuming method from the set of attacks we used, we tried to use it for adversarial training. We experimentally showed that learning model against FGSM examples increased accuracy against FGSM but did not yield model robust against other methods.

After this discovery we used as a reference attack PGD. We expected the model should not overfit to this samples because the perturbations are more complex and harder to learn. This assumption was successfully demonstrated and will be described in the following text.

## 5.1 Similarity level of adversarial examples

Finding adversarial examples is a constrained optimization problem. The goal of optimization is to generate such samples that the target model will not be able to correctly classify, while they obviously belong to certain class. The constraint for this problem is that the

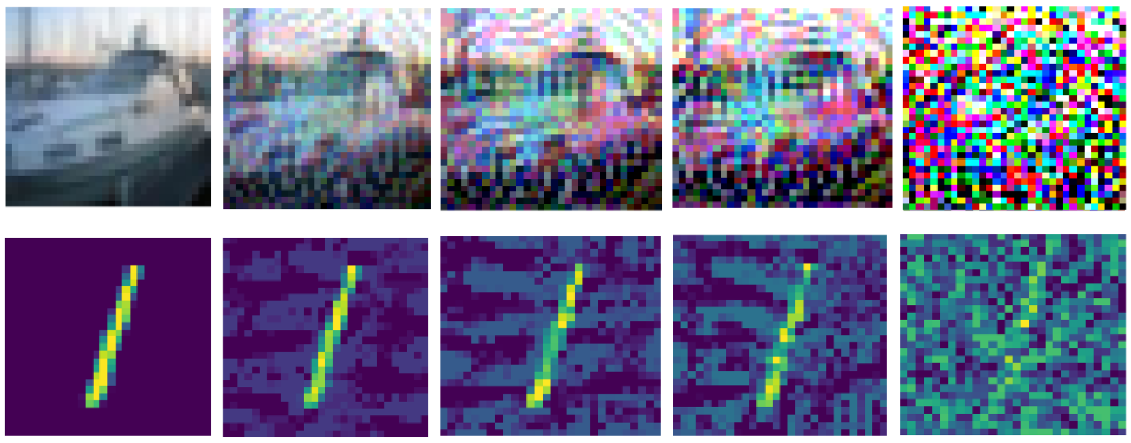
Figure 5.1: Natural samples and samples perturbed with 0.1, 0.2, 0.3 and 1.2-$\epsilon$ in this order. Higher located images are from CIFAR10 dataset and the lower ones are from MNIST. The figure illustrates how different are perturbed images from the original one wrt. epsilon. Norm used for generating perturbation is $L_\infty$.

adversarial examples must satisfy certain level of similarity to the original samples. The similarity is in this work determined by the value of epsilon wrt. norm.

First step for generating suitable adversarial examples is choosing suitable value for the epsilon. Epsilon value depends mainly on dataset but also on required level of similarity, e.g. sample could be recognizable with distance $\epsilon = 0.5$, however it is ugly and unrealistic so we choose $\epsilon = 0.1$. It is possible that attack with higher epsilon would not decrease accuracy of the network significantly more than one with lower epsilon. However, training against high-epsilon attack may be impossible and would not make sense if the samples would be strongly different to original samples.

This section is reasoning selected value of $\epsilon$. Correctly selected epsilon wrt. to norm should satisfy successful training and robustness against all samples similar to original ones.

As mentioned before, important factor for choosing $\epsilon$ value for attacks is strong visual similarity of generated adversarial image with an original one. Such $\epsilon$ differs according to dataset and also used norm. For all experiments we will use norm $L_\infty$. As we can see in the figure 5.1 perturbing *CIFAR10* image by more than $\epsilon = 0.1$ yields too unclear image, while perturbing *MNIST* image by more than $\epsilon = 0.3$ makes image too unrealistic. Therefore we decided to use epsilon for *CIFAR10* $\epsilon = 0.1$ and for *MNIST* $\epsilon = 0.3$.

## 5.2 Evaluation of adversarial methods

The motivation of our work is discovering and validating the solution against vulnerability of neural networks to adversarial examples. For this reason we need to demonstrate that good working neural network model fails against adversarial attacks.

This experiment shows accuracy of naturally trained models for both MNIST and CIFAR10 datasets against attacks in comparison to accuracy evaluated on natural data. In the tables 5.1 and 5.2 we can see how sensitive are the neural network models to adversarial attacks wrt. value of epsilon. It makes sense just to use such epsilon value in the context of our work, that modifies the image just the way a humans eye could recognize it. That is also the reason why we tested values of epsilon 0.1, 0.2 and 0.3 for both datasets (see Figure 5.1).

To specify experiment details, both models were trained with the 10 epochs. For evaluation we used testing set of 10 000 samples. Accuracy evaluated against some attack means in this work percentage of correctly classified adversarial examples generated by specific attack. Every samples $X_i^{adv}$ from the adversarial dataset is generated by perturbing sample $X_i$ from the original testing set at most by epsilon, thus there is a bijection between original testing dataset and adversarial dataset. Epsilon is an important parameter for single step methods like FGSM, however, for multi-step methods we also consider step size $\gamma$ and number of attack iterations. Iteration or a step, is a process of perturbing the original image at most by $\gamma$ (for FGSM $\gamma = \epsilon$).

Notice, that MNIST model is keeping relatively high accuracy for a small value of epsilon, however with an epsilon of 0.2 and higher, the accuracy is significantly decreased. This could be caused by the low complexity of MNIST samples. For comprehension we could say that naturally trained MNIST model c-robust wrt $\epsilon = 0.1$ and all used attacks, where $c = 0.8426$, because the accuracy of classifying natural data minus accuracy of classifying worst adversarial data is 84.26%. For greater epsilons is robustness strength even lower.

Another intriguing property of the MNIST model is that it keeps a high accuracy against FGSM attack comparing to other attacks. Even we expected that FGSM will be the least successful attack, the difference to other attacks is significant. This observations shows us that local error maximums in $\epsilon - balls$ around original samples are high and steep, thus for finding them we need iterations. On the contrary, results from attacks to CIFAR10 are more stable and local maximums are less steep and the area is more consequent (less value of derivations).

As we can also see, attacking time does not differ wrt. epsilon value but more likely on number of steps, which is the same for all (multi-step) attacks. Notice that all multi-step methods last slightly the same time. FGSM, which is single-step method should take approximately as much time as one step of PGD, in our case it would be time needed for PGD divided by 12 (since we used 12 iterations). When you check values in Tables 5.1 and 5.2 you can see that the previous statement is approximately valid in most cases.

Evaluating adversarial methods for CIFAR10 model shows that accuracy differs wrt. epsilon less than in case of MNIST, in some cases the accuracy is even better for greater epsilon value. FGSM decreases accuracy the least, as we expected, however the accuracy is not as different to multi-step methods as for MNIST dataset. This is probably caused by greater complexity of CIFAR10 data comparing to MNIST, and the model is less overfitting to natural samples. However, this observation does not solve the problem of adversarial examples, since the accuracy evaluated on malignant data is still worse than random prediction model, even it is 10 time preciser than for MNIST in some cases.

### Local error maximums found by PGD

Since PGD is the only method in our stack of attacks uses random starting point, we are wondering if the results achieved by this method are not unstable, since it could distort final result. As the loss function of the target model could contain few local maximums, we should consider possibility that PGD could yield radically different results for more restarts of the algorithm. Notice, that this phenomenon is only possible for non-deterministic methods.

In 3.2 we adapted hypothesis from Madry et al. [11], that starting point of PGD is irrelevant and all local maximums found by PGD are almost the same. We demonstrated

| Attack | attack time | accuracy | epsilon |
|---|---|---|---|
| no attack | 0s | 99.2% | 0.0 |
| MIM | 85.24s | 25.2% | 0.1 |
| BIM | 81.55s | 14.94% | 0.1 |
| PGD | 83.44s | 15.77% | 0.1 |
| FGSM | 11.32s | 74.58% | 0.1 |
| MIM | 81.27s | 0.87% | 0.2 |
| BIM | 80.53s | 0.74% | 0.2 |
| PGD | 79.49s | 0.94% | 0.2 |
| FGSM | 9.69s | 31.65% | 0.2 |
| MIM | 84.70s | 0.57% | 0.3 |
| BIM | 82.20s | 0.55% | 0.3 |
| PGD | 81.50s | 0.44% | 0.3 |
| FGSM | 7.48s | 14.75% | 0.3 |

Table 5.1: Demonstration of how every single attack deceases accuracy of the **MNIST** model trained on 10 epochs wrt. epsilon. The attack time describes time needed for generating 10 000 samples. All iterative attacks ran with 12 iterations and the step size $\gamma$ was the current epsilon divided by 6, thus the total distance of the gradient descent was twice the epsilon. This should take the algorithm a chance to find any local error maximum in the allowed distance from the original sample. Norm used for the perturbations was $L^{\infty}$.

| Attack | attack time | accuracy | epsilon |
|---|---|---|---|
| no attack | 0s | 75.56% | 0.0 |
| MIM | 234.85s | 7.94% | 0.1 |
| BIM | 230.40s | 8.42% | 0.1 |
| PGD | 227.80s | 4.21% | 0.1 |
| FGSM | 20.44s | 10.81% | 0.1 |
| MIM | 224.9s | 5.84% | 0.2 |
| BIM | 231.3s | 6.19% | 0.2 |
| PGD | 230.15s | 2.43% | 0.2 |
| FGSM | 20.27s | 11.42% | 0.2 |
| MIM | 231.89s | 4.19% | 0.3 |
| BIM | 235.54s | 4.45% | 0.3 |
| PGD | 229.81s | 1.8% | 0.3 |
| FGSM | 20.47s | 8.9% | 0.3 |

Table 5.2: Demonstration of how every single attack deceases accuracy of **CIFAR10** model trained on 10 epochs wrt. epsilon. The attack time describes time needed for generating 10 000 samples. The step size and iterations number were used same way as in Table 5.1.

| MNIST accuracy | CIFAR10 accuracy | Iterations count |
|---|---|---|
| 36.64% | 3.97% | 5 |
| 9.78% | 3.92% | 10 |
| 7.05% | 3.84% | 15 |
| 5% | 3.99% | 20 |
| 3.35% | 3.9% | 25 |
| 2.52% | 3.9% | 30 |
| 1.83% | 3.86% | 35 |

Table 5.3: Progress of PGD with fixed step size $\gamma = 0.01$, epsilon for MNIST 0.3 and for CIFAR10 0.1. Both models are trained on 20 epochs.

this statement by running PGD with 20 restarts against MNIST and CIFAR10 models with same architecture trained on 10 epochs. For MNIST dataset the average accuracy was 0.45% and standard deviation was only 0.0317%, thus there was not any PGD run with significantly better results. Experiment with CIFAR10 also approved our hypothesis with an average accuracy 4.656% and and standard deviation 0.93%.

In our work we rely on the assumption that adversarial samples found by PGD cannot be improved by restarting the method.

**Evaluation of attacking step size for PGD**

Step size is one of the important parameters for multi-step methods. For single-step methods is step size commonly equal to epsilon. We maximize the epsilon distance from the original image because we assume that the bigger distance from an original samples, the harder it is to recognize the image for target model and the greater local loss maximum could be found.

Beware that this statement is not formally correct and the highest local maximum (or just highest point) of the epsilon-ball around the original sample does not necessarily lie on the border of the ball. However, we rely on the assumption that the greater distance from the original samples is, the less similar samples there lie and there is a greater probability that the target model fails.

In this experiment we evaluate various number of attack steps of PGD with fixed size of step $\gamma = 0.05$. After certain number of steps the loss function of target model should stop increasing and the attack probably reached optimal result. The number of steps differs according to epsilon, dataset and step size. We applied this experiment for both MNIST and CIFAR10. In the Table 5.3 we can see that increasing attack iterations decreases accuracy on MNIST but does not affect CIFAR10 model. This could be caused by the fact that epsilon of MNIST if 3 - times larger that for CIFAR10 model. This experiment shows that PGD step size matters and it is required to experiment with this parameter.

## 5.3 Filters against adversarial examples

As we proposed in Section 4.1, filters are often used also for reducing noise in the samples. Since we consider adversarial sample as a sample with added noise, we experimentally show how could filters be helpful for classifying perturbed samples. This section demonstrates usage of the filters as a defence method against adversarial examples for two datasets -

| Dataset | Natural | Adversarial | Thresholded adversarial |
|---|---|---|---|
| MNIST | 99.2% | 0.5% | 94.4% |
| CIFAR10 | 74.4% | 3.4% | 9.6% |

Table 5.4: Demonstration of adversarial accuracy against thresholded samples in comparison to adversarial and natural samples. Attacks were performed using PGD with $\epsilon = 0.3$ for MNIST and $\epsilon = 0.1$ for CIFAR10.

| Dataset | Natural | Adversarial | Median filtered adversarial |
|---|---|---|---|
| MNIST | 99.2% | 0.5% | 0.93% |
| CIFAR10 | 74.4% | 3.4% | 20.4% |

Table 5.5: Demonstration of effectiveness of median filter against adversarial attacks. Attacks were performed using PGD with $\epsilon = 0.3$ for MNIST and $\epsilon = 0.1$ for CIFAR10.

*MNIST* and *CIFAR10*. Filters are often part of a neural networks architecture, however we will use them here as a „secret“ first layer even against white box attacks.

### 5.3.1 Thresholding

This simple filter sets value of color channel for each pixel to zero if it is under the value of threshold, respectively to maximal value (in 8 bit color system 255) otherwise (see Subsection 4.1.1). This experiment achieved impressive results for *MNIST* dataset, since after applying thresholding the accuracy increased from 0.5% to 94.4% with linear time complexity wrt. dataset size. This would be acceptable solution in terms of both accuracy and time. Table 5.4 shows that using thresholding could be really effective for simple datasets like *MNIST*. On the contrary, for more complex datasets like *CIFAR10* it yields unrecognizable images even for human and increases success just a little bit. As we can see in the figure 5.2 thresholding seems to be visually helpful for *MNIST* data, data from *CIFAR10* are after applying threshold filter unrecognizable. Notice that although the improvement for CIFAR10 model with thresholding is not satisfying our needs, it is intriguing that even if it yields image unrecognizable for human, it increases neural network accuracy of the target model.

### 5.3.2 Median filter

Since the thresholding that we evaluated in subsection 5.3.1 produces too drastic changes of source images we decided to use median filter. It is meant to reduce noise added by adversarial methods as explained in 4.1.2. We evaluated this method using same samples as in previous experiment and the interesting observation is that this method yields quite better results for CIFAR10 model comparing to thresholding but the achieved result 20.4% against filtered adversarial examples is still not satisfying our requirements. For MNIST model median filtering produces just tiny increasing of accuracy. For visual comparison see 5.2. Results of experiment with median filter noted in Table 4.1.2.

## 5.4 Adversarial training

Training network robust against adversarial examples requires iterative process of attacking the model and generating samples. This samples are used for learning perturbations of

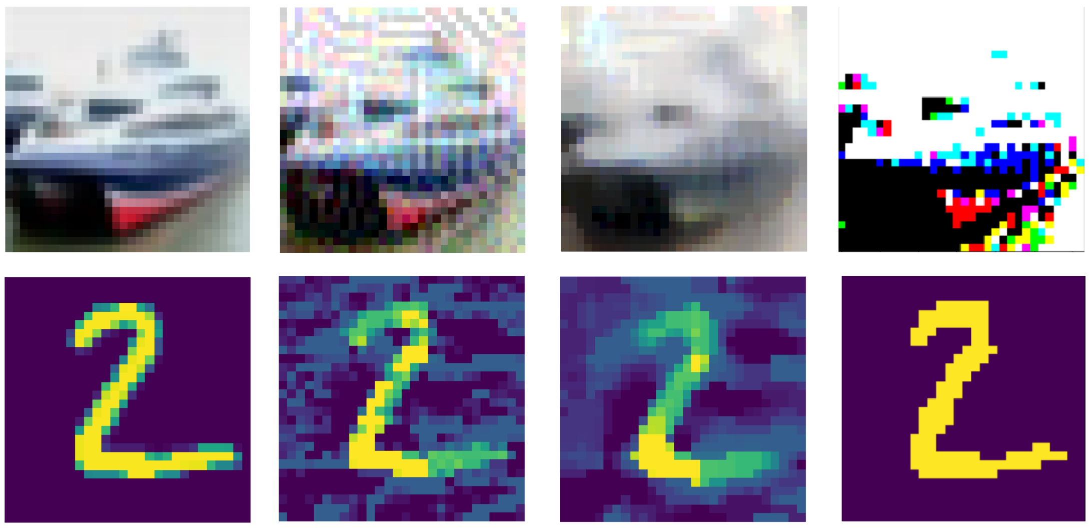

Figure 5.2: Natural, adversarial, adversarial filtered data using median filter and adversarial filtered data using thresholding in this order.

attack strategies. Method for generating adversarial samples, so called *target attack*, needs to be chose properly. Some perturbations could be easy to learn and could cause overfitting. This phenomenon happens when we train against one-step examples, specially generated by FGSM [11]. We demonstrate this hypothesis in subsection 5.4.2. Another target attack that we used is Projected Gradient Descent (see subsection 5.4.3). For both experiments we used *MNIST* dataset. Except for attacking methods we decided to use data augmentation to extend dataset with hypothesis of increasing generalisation of neural network.

### 5.4.1 Training against augmented samples

As explained in section 4.2.1, data augmentation is a method for increasing the diversity of the data and extending dataset. It should make models prediction invariant wrt. rotation, scale, etc., and as we hope also small perturbations. There is an idea that extending diversity of training data could yield more robust models against adversarial examples. We experimentally evaluated data augmentation and find out that this method is not improving c-robustness sufficiently. As you can see in Table 5.6, we observed small improvements of accuracy evaluated on adversarial examples for both MNIST and CIFAR models but the final adversarial accuracy did not even reach accuracy 10% which is theoretical accuracy of random classifier for 10 classes. We also observed some decrease of accuracy for natural data with augmentation. This may be caused by the fact that bot datasets are large enough and evaluation data are similar to training data. This exploration leaded us to last defence method of our research explained in Subsection 4.2.2.

### 5.4.2 Model trained on FGSM perturbed MNIST data

As we stated earlier it is faster to train model against single-step as FGSM method than against PGD. Experiment described in this subsection shows results of MNIST model trained exclusively on perturbed data from MNIST dataset with FGSM as a reference attack.

| Model | Attack time | PGD accuracy | Natural accuracy |
|---|---|---|---|
| MNIST | 140.26s | 0.4% | 99.4% |
| MNIST augmented | 145.36s | 0.51% | 99.1% |
| CIFAR10 | 190.22s | 3% | 82.4% |
| CIFAR10 augmented | 189.63s | 6.9% | 80.4% |

Table 5.6: The table demonstrates effect of data augmentation used for extending training dataset to robustness against adversarial examples. We noticed only poor improvements of the models robustness.

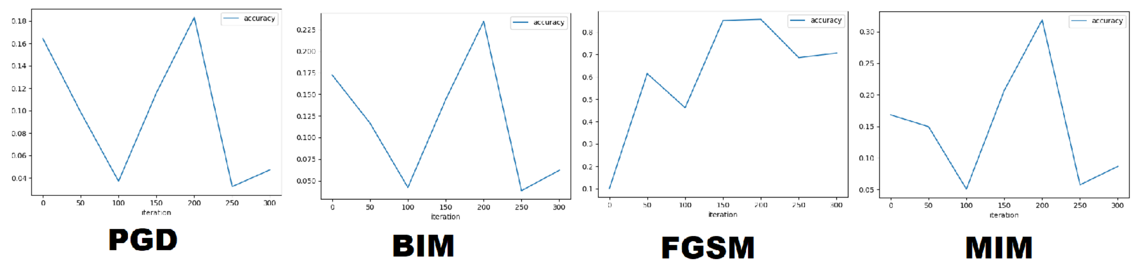

Figure 5.3: We can see the progress of models accuracy training on FGSM samples.

Highest accuracy reached by this model against PGD attack was $\sim$ 19% after 200 training iterations. One the other hand, the method achieved outstanding results against FGSM attack. After 200 iterations model achieved accuracy $\sim$ 85% (see Figure 5.3). As we can see in the Table 5.7, training against FGSM samples improved c-robustness and result accuracy is definitely better than accuracy of random classifier for 10 classes which is 10%, thus such result worths to study.

Another interesting phenomenon is that accuracy does not converge along with iterations to 100% but we need to select the optimal number of iterations for best results. This divergence is undesirable in the training process and may indicate problems with this method for future use.

The result is better comparing to no adversarial training, however still not sufficient. Note that trained model can handle FGSM attack quite well - this is probably reason of overfitting. The perturbations generated by FGSM are simple and easy to learn [11]. For better performance and to avoid overfitting we need a reference attack that yields more sophisticated perturbations. For this purpose we employed PGD as a reference attack.

### 5.4.3 Adversarial training with PGD reference attack

Performing adversarial training against samples generated by PGD was the last option because its time complexity. Since we generated adversarial examples with 12 steps of PGD it would take approximately 12-times more than FGSM.

However, the adversarial training with PGD as a reference attack was successful and as we can see in the Table 5.7, the training yield 0.06-robust model wrt. $\epsilon = 0.3$ for MNIST dataset. Or in other words, the most successful training could decrease models accuracy only by 6%.

PGD is more sophisticated and also more time consuming method. Applying this method may require few days of training for performing about 2000 training iterations for MNIST model. As you can see in the Figure 5.4, accuracy is constantly increasing along

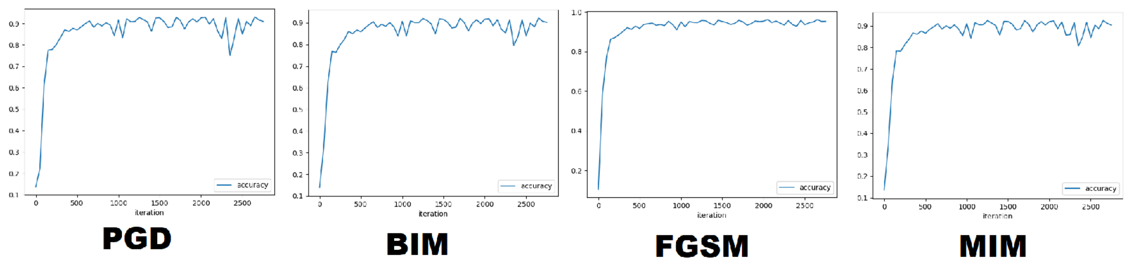

Figure 5.4: Demonstration of models accuracy progress during training iterations of adversarial training against PGD.

| Reference attack | no attack | FGSM | PGD | BIM | MIM | AdvGAN |
|---|---|---|---|---|---|---|
| FGSM | 96.1% | 92.5% | 18.2% | 23.5% | 33.9% | 50.05% |
| PGD | 98.5% | 96.1% | 93.6% | 92.5% | 93.9 % | 94.26% |
| no attack | 99.2% | 14.75% | 0.44% | 0.55% | 0.57% | 9.85% |

Table 5.7: Demonstration of adversarially trained MNIST network with two reference attacks wrt. $\epsilon = 0.3$. All multi-step attacks were performed with 12 steps and step size $\lambda = 0.05$. Results for model trained with PGD reference attack are measured after 2000 training iterations and with FGSM reference attack after 200 training iterations.

with iterations and it makes the training process more stable in comparison to FGSM as a reference attack.

Such a good results in terms of robustness for MNIST model could be also achieved by thresholding (see 5.3.1). An advantage of thresholding is no need of computationally hard adversarial training. On the contrary, once the robust model is trained there is no need to filter an input images. Another important issue of thresholding is that it only works for some models (e.g. MNIST). Our goal is to provide method for improving robustness of any neural network model.

For evaluation of adversarial training with PGD as a reference attack we used also CIFAR10 dataset. Our experiments showed that this method yields also CIFAR10 model with c-robustness that satisfies our requirements. The difference from training MNIST robust model is that CIFAR10 requires enormously larger number of iterations and every iteration consumed longer time. CIFAR10 adversarial training reached its best (stopped improving) after 80000 iterations which is comparing to 2000 iterations needed for MNIST really huge time-resources consumption. The question for consequent research is if it is feasible to perform adversarial training for model that classifies much more complex dataset.

| Reference attack | no attack | FGSM | PGD | BIM | MIM | AdvGAN |
|---|---|---|---|---|---|---|
| PGD | 80.5% | 61.81% | 55.46% | 56.23% | 58.48% | 69.45% |
| no attack | 82.4% | 10.81% | 4.21% | 8.42% | 7.94% | 12.59% |

Table 5.8: Demonstration of adversarially trained CIFAR10 network with PGD as a reference attacks wrt. $\epsilon = 0.1$. All multi-step attacks were performed with 12 steps as in Table 5.7 and step size is also epsilon divided by 6 thus $\lambda = 0.0167$. Results for model trained with PGD reference attack are measured after 80000 training iterations. We skipped training on FGSM samples because experiments with MNIST indicate it will not yield required results.

Chapter 6

# Conclusion

The goal of this work was to demonstrate that adversarial examples are able to cause significant danger for neural network applications and also to propose, implement and evaluate solution for this issue. This work focused on optimization approach of improving the robustness in multi-class image classification problems. The field of ML security is certainly much larger and even adversarial examples could be generated also for datasets containing voice, malware, etc.

We demonstrated some basic adversarial attacks specifically FGSM, PGD, MIM, BIM and AdvGAN. Every attack except for AdvGAN is white-box attack. This focus on white-box attacks is reasonable - all white-box attacks could be use also as a black-box attack because of transferability phenomenon (see Section 2.3.1) thus it is important to study carefully these attacks. Notice, that white-box attacks used in this work are based on the same principle of gradient descent optimization. PGD extends FGSM, BIM is a special case of PGD and MIM extends PGD. There is plenty of other white-box attacks but it is valid to say that vast majority of state-of-the-art attacks are extension or modification of attacks described in this work.

Our work also described approaches for solving the problem of adversarial examples and creating robust model against it. We showed that deeper neural network models with greater capacity have better c-robustness with higher accuracy. This observation included that better architecture could improve c-robustness, while keeping same or better natural data accuracy but improving architecture alone does not yield model with desired robustness. Since adversarial training requires to learn much more information than standard neural network training, number of trainable parameters is also important and should be sufficient wrt. data size and epsilon. Deeper research about dependency of robustness and architecture should be performed in the future work and we believe it is also a way how to face attacks. Another possibility how to increase success against adversarial examples is to use filters. Even simple filter as thresholding preformed outstanding results for classifying MNIST adversarial examples. Smoothing samples and reducing noise using filters could be a great defence, problem is that some filters could improve accuracy for adversarial data but also decrease accuracy for natural data. In fact, because of image datasets diversity it may be impossible to find one general filter working well for all or at least wide range of image classification problems. Preprocessing data for both training and evaluation using neural network is definitely promising method for improving robustness and should be also studied in detail in the future work. Our final approach was the robust optimization where we were solving saddle point (or minimax) problem. The term minimax means that we were trying to minimize maximized loss of the target model. Loss maximization was performed by some of previously proposed adversarial attacks. The challenging part was to choose suitable attack for maximization the loss function that generates samples accordingly complex samples. The first idea was to use FGSM as a reference attack and repeatedly train the target model on batches generated by FGSM. In result, we obtained MNIST model with good performance against FGSM samples without significant loss of natural accuracy. Accuracy of model on natural MNIST data was 96.1% and on adversarial FGSM data 92.5% thus for only FGSM attack the model was 0.036-robust wrt. $\epsilon = 0.3$ what would be acceptable result. Evaluation of this model using multi-step attacks showed us that model is not robust enough to face e.g. PGD where it achieved only 18.2%. The model seemed to overfit to FGSM samples, while performing well on natural data. Notice, that FGSM always yields only samples on the border of epsilon-ball around the original image ignoring any other malignant samples closer to original samples. There definitely is plenty of other samples within the epsilon-ball that fool the target model. For learning the model also this samples we needed some reference attack that find also such samples. This observations leaded us to use PGD as a reference attack because PGD generates more complex data and is able to find samples with high loss values anywhere within the epsilon-ball. The target MNIST model trained on PGD samples also kept high accuracy on natural data, while performing very well against any attack described in this work. The most harmfull attack for model trained on PGD samples was BIM and the accuracy on BIM samples was 92.5%. As the accuracy on natural data was 98.5% the model was 0.06-robust wrt. attacks described in this work what is an outstanding result. For further evaluation of method we also trained CIFAR10 model on PGD samples. The training process took enormously larger amount of time, since we needed about 80000 iterations for training the reaching the optimal model (accuracy on both adversarial and natural data stopped improving with iterations). Final accuracy of model trained on PGD samples evaluated on natural data is 80.5% which is not significant decrease comparing to naturally trained models accuracy 82.4%. Since PGD attack was able to decrease accuracy to 55.46% the model is 0.2504-robust wrt $\epsilon = 0.1$. This result is not as successful as in case of MNIST, however, we need to consider that CIFAR10 dataset is much more complicated than MNIST.

Even we showed that we can face first-order adversaries, which are currently state-of-the-art attacking methods, the research is still not done yet. There exist also better adversarial attacks than we described - commonly some modification of attacks we used. One of them is for examples PGD attack with output diversified initialization [17] that could decrease accuracy of robust model under 50% according to experiments in Madry Challenge [11]. We should also consider possibility of new attack strategies, especially higher-order methods. Bringing this method could completely remove our advantage of relatively robust model. We believe that research of higher-order attack method should be the part of the future study of the field of ML security. Another weakness of our research is that we assume some constant value of $\epsilon$, however there exist samples with high distance from original data, but looks similar, therefore there is still possibility to easily generate adversarial examples with higher $\epsilon$, which look pretty similar to natural data and fool the model this way. Very challenging and also important would be the research of measuring similarity between samples and possibility of finding some kind of border between similar and different samples. This work discussed and studied only case of adversarial examples for image data. The future work should focus on voice (speech, sound) data. This days large amount of malware is detected using machine learning and it would be really interesting and also challenging to find some ways how enemies could generate malware adversarial samples and try to find ways of defence.

# Bibliography

[1] Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In: IEEE. *2017 ieee symposium on security and privacy (sp)*. 2017.

[2] Dong, Y., Liao, F., Pang, T., Hu, X. and Zhu, J. Discovering adversarial examples with momentum. *ArXiv preprint arXiv:1710.06081*. 2017.

[3] Goodfellow, I., Pouget Abadie, J., Mirza, M., Xu, B., Warde Farley, D. et al. Generative adversarial nets. In: *Advances in neural information processing systems*. 2014, p. 2672–2680.

[4] Goodfellow, I. J., Shlens, J. and Szegedy, C. *EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES*. ICLR 2015, 2015. Retrieved from: https://arxiv.org/pdf/1412.6572.pdf.

[5] Haohui. *Adversarial Attacks in Machine Learning and How to Defend Against Them* [online]. https://towardsdatascience.com/, december 2019 [cit. 2020.01.20]. Available at: https://towardsdatascience.com/adversarial-attacks-in-machine-learning-and-how-to-defend-against-them-a2beed95f49c.

[6] Huang, X., Kwiatkowska, M., Wang, S. and Wu, M. Safety verification of deep neural networks. In: Springer. *International Conference on Computer Aided Verification*. 2017, p. 3–29.

[7] Kitano, H. Biological robustness. *Nature Reviews Genetics*. Nature Publishing Group. 2004, vol. 5, no. 11, p. 826–837.

[8] Knagg, O. Know your enemy: Why adversarial examples are more important than you realize. 2019. Available at: https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3.

[9] Kurakin, A., Goodfellow, I. and Bengio, S. Adversarial examples in the physical world. *ArXiv preprint arXiv:1607.02533*. 2016.

[10] Kurd, Z., Kelly, T. and Austin, J. Developing artificial neural networks for safety critical systems. *Neural Computing and Applications*. 2006, vol. 16, no. 1. DOI: 10.1007/s00521-006-0039-9.

[11] Madry, A., Makelov, A., Schmidt, L., Tsipras, D. and Vladu, A. Towards deep learning models resistant to adversarial attacks. *ArXiv preprint arXiv:1706.06083*. 2017.

[12] Papernot, N., Carlini, N., Goodfellow, I., Feinman, R., Faghri, F. et al. Cleverhans v2. 0.0: an adversarial machine learning library. *ArXiv preprint arXiv:1610.00768*. 2016.

[13] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B. et al. Practical black-box attacks against machine learning. In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, p. 506–519.

[14] Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*. Elsevier. 1964, vol. 4, no. 5, p. 1–17.

[15] Shokri, R., Stronati, M., Song, C. and Shmatikov, V. Membership inference attacks against machine learning models. In: IEEE. *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, p. 3–18.

[16] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D. et al. Intriguing properties of neural networks. *ArXiv preprint arXiv:1312.6199*. 2013.

[17] Tashiro, Y., Song, Y. and Ermon, S. Output Diversified Initialization for Adversarial Attacks. *ArXiv preprint arXiv:2003.06878*. 2020.

[18] Xiao, C., Li, B., Zhu, J.-Y., He, W., Liu, M. et al. Generating adversarial examples with adversarial networks. *ArXiv preprint arXiv:1801.02610*. 2018.