BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

Improving robustness of neural networks against adversarial examples

Martin Gaňo*

Abstract
The main goal of this work is to design and implement the framework that yields robust neural network model against whatever adversarial attack, while result models accuracy is not significantly lower comparing to naturally trained model. Our approach is to minimize maximization the loss function of the target model. Related work and our experiments lead us to the usage of Projected gradient descent method as a reference attack, therefore, we train against data generated by PGD. As a result, using the framework we can reach accuracy more than 90% against sophisticated adversarial attacks on MNIST dataset. The greatest contribution of this work is an implementation of adversarial attacks and defences against them because there misses any public implementation.

Keywords: Neural networks — Optimization — Machine learning

Supplementary Material: Downloadable Code

*xganom00@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Professionals are worried for a long time about using neural networks in safety-critical applications. Safety-critical systems are employed in many areas including transport industries, medicine and defence [1], for instance, autonomous cars or operating robots. The reason of concern is the fact that neural networks contain an enormous number of trainable parameters and that's why they are too complex for human comprehension, in other words, they are so-called black-boxes for humans. Because of that, it is infeasible to explain or fix a misclassification, or another kind of error caused by seemingly faultless model. A significant danger comes with simple methods for creating adversarial examples. An adversarial example is an almost indistinguishable input from the original sample, however, it is misclassified [2]. This way can enemy affect the result of model prediction (breaking face detection, fooling autonomous car, etc...) and manipulate the target application. In some extreme cases, this kind of attack can result in fatal consequences like serious endangerment humans health, life or property. It will never be possible to resist all kinds of attacks used by enemies. They keep improving their criminal 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 machine learning models.

The main contribution of this work is the framework that enables to perform adversarial training for any neural network architecture with any dataset and that is important for experimenting with adversarial attacks and defences to make progress in this field of research. There's a strong need for such a tool because there's no available public implementation of defence methods for general usage. We must mention frame-

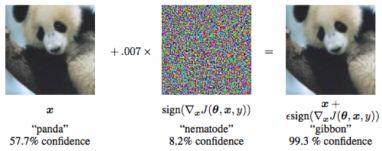

Figure 1. Little perturbations can cause misclassification of a model with high confidence. Adapted from [2].

work *Cleverhans* with implemented attacks, however, this work currently misses defence methods. Madry et al. provided an implementation of adversarial training for some concrete model both for CIFAR10 and MNIST datasets [3] but this is also not sufficient for ML community and it's the reason why we are presenting the general implementation of adversarial training for any model in Python language using high-level library *Keras*.

Notice, that the defence against adversarial attacks (or adversaries) differs according to a type of attack. Related work and our experiments gave us hypothesis, that we can train a model that is robust to all attacks (wrt. the distance between adversarial example and natural sample), which are using only *first-order* information [3]. In other words, we are now able to create model resistant to the whole class of attacks. We have experimentally proven the hypothesis for few white-box attacks implemented by Ian Goodfellow et al.[1] and our implementation of Generative adversarial network designed for generating adversarial samples called AdvGAN [4]. All these attacks were not able to decrease the network's accuracy under 90%. There are no other methods of attack that *first-order* used in this work.

2. Background

This section contains definitions and explanations of important terms used in this paper.

2.1 Adversarial example

Informally, it is a sample, which is almost indistinguishable from the original sample for human, while the model misclassified it. A valid explanation is also that it is a little perturbed original data, that are misclassified by the target model. We need also a formal and unified definition of adversarial example. We need a formal definition of adversarial example.

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

Typical metrics used in previous work are L^∞ and L^2 distances since they properly simulate similarity for humans [3].

L^∞ metric, sometimes called *Chebyshev distance* between two vectors is defined as the greatest value of their differences along any coordinate dimension.

$$d_{chebyshev}(x,y) = \max_i(abs(x_i,y_i)) \tag{1}$$

L^2 metrics is also called Euclidean distance.

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

where n is the dimension of the vector.

2.2 Robustness

The general definition of robustness is the ability of a system to handle perturbations or errors that might affect it without changing the initial configuration.

There's a need to adapt this definition for the problem of robust optimization. Theoretically, it would be really simple to define robustness using the definition of adversarial examples this way: *The model is robust if it is not possible to generate any adversarial example*. This definition would be useless in most practical cases because only models with 100% accuracy are robust according to the mentioned definition. Our definition must cover this case, but also must to be much weaker to make it feasible to realize it.

Definition 2 *An optimal model is robust if it stays optimal under any allowed perturbation of input data.*

Definition 2 is valid but we need a more specific one for the final evaluation of robustness.

Definition 3 *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 $d(X_i^{adv},X_j^{nat}) < \varepsilon$, where d is some metric and ε is small constant distance. Then t is robust if $a_{nat} - a_{adv} < c$, where c is constant value.*

2.3 Transferability phenomenon

Many black-box methods are based on so-called *transferability phenomenon*. According to [5] are adversarial examples transferable between neural network architectures i.e. black-box attacks often rely on that

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

generating malignant samples against the target model will be also harmful to another one.

According to Madry et al. [3] it is also useful for the attacker to choose suitable model because training only against strongest models doesn't guarantee that the adversarial samples will be most transferable as they could.

2.4 Adversarial attack

Generally, we understand finding adversarial examples as a *constrained optimization problem*. For this paper, we are defining adversarial attacks as functions with a variable amount of parameters and the function returns adversarial examples (see the subsection 2.1). The number of attack parameters differs according to the type of attack and used method. It is common for attacks to take as an argument real data with their labels - this is an essential variable for any kind of attack. Sometimes the original data is used for training adversarial models (e.g. Adversarial generative networks) or in other cases the output data are deriving directly from the original data (e.g. Fast gradient sign method). Another possible parameter is the target model. This is often (but not necessary) used, because we could design our target model and expect that the real target model will behave similar way (see 2.3). We distinguish two types of adversarial methods, according to information about the model that is passed as an argument. If adversarial functions don't access to target models parameters, the method is called **black-box**. The second type is called **white-box** attacks. We consider attack as a white-box when the enemy has access to all model parameters and the whole model structure.

3. Related work

Szegedy at al. in 2014 [6] demonstrated few properties of neural networks, including but not limited to fact that one adversarial sample is often misclassified by few models, neural networks are vulnerable to adversarial examples and interesting phenomenon that some adversarial examples (for some dataset) are indistinguishable for human. Our understanding danger of adversarial examples in neural networks is based on these observations.

After that Ian Goodfellow et al. explained adversarial examples in [2]. 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 our extended implementation of these methods is an important part of the framework presented in this work.

Framework for safety verification of neural networks was proposed by Huang et al. [7]. The framework guarantees to find all existing adversarial examples by exhaustive search around the region of original sample implemented by technologies like Z3.

Research by Kurakin et al. [8] brought us a few ideas about adversarial training including a summary of attacks. They demonstrated the transferability phenomenon, label leaking and difference between adversarial training using one-step attacks and iterative methods (with many steps). This will be explained later in this paper.

We adapted approach to adversarial training introduced by a group of scientist from MIT in their paper [3]. In this work, they proposed hypothesis, that if the 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 my work against several adversarial attacks. The mentioned paper also studies the relationship 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 89% accuracy against state-of-the-art attacks (on MNIST data).

The method that attacked most successfully robust model trained using defence by Madry is extended implementation of Generative adversarial networks (GAN) [4]. GAN was proposed by Ian Goodfellow et al. [9]. Authors created a framework called *AdvGAN* for perturbing original data to adversarial examples.

4. Adversarial methods

The term Adversarial attack was explained in subsection 2.4 and we are using terms adversarial method and adversarial attack with the same meaning. In this section, I will describe the methods used for creating adversarial examples and evaluating the robustness of the target model. All methods use only *first-order* information.

4.1 Fast gradient sign method

The method was firstly proposed 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. The formula for computing

result sample is

$$X^{adv} = X + \varepsilon * sign(\nabla_x L(X,Y)) \quad (3)$$

where X^{adv} is adversarial sample, X is original sample, ε 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 computation time because it is computed just with one single step. [8]

4.2 Basic iterative method

The basic iterative method is a straightforward extension of FGSM 4.1, that is applied in iterations with small step size and we need to clip pixel values after each step - this ensures that they are in ε-ball of original data sample [10]. As X_n^{adv} we denote adversarial example computing with n iterations. To compute X_n^{adv} we first need to set X_0^{adv} as follows:a

$$X_0^{adv} = X \quad (4)$$

and to compute X_n^{adv} we use this formula:

$$X_n^{adv} = clip_{X,\varepsilon}\{X_{n-1}^{adv} + \alpha * sign(\nabla_x L(X_{n-1}^{adv}, Y)\} \quad (5)$$

where $sign(x)$ is function with same meaning as in Equation 4.1 and $clip_{X,\varepsilon}(x^{pert})$ is a function that returns x^{pert} if $D(X, x^{pert}) < \varepsilon$, otherwise it returns nearest point to x^{pert} that is within ε-ball around X.

This method also requires access to model parameters, so we consider it as a white-box attack.

4.3 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 the model robustness, it will be described more detailed and with special attention. Similarly, as methods mentioned above, we are using PGD to find such adversarial inputs, that maximize the loss of the model while the distance between generated data and original data according to certain metric does not exceed ε. Correctly chosen ε (with the correctly chosen norm) can guarantee, that generated examples will be similar, or even indistinguishable, or at least imperceptibly different to original samples.

The algorithm is an extension of both FGSM and Basic iterative method. We first set X_0^{adv} to random point within the L^p ball (in our case $p = \infty$) with radius of ε same way as in equation 4. Then we make a gradient step with size α in the direction of greatest loss and project back to L^p ball [2] (clip) as in equation 5. Then we apply equation 5 until convergence, i.e. until the value of loss function slows down increasing from step to step [11] as you can see in the figure 2.

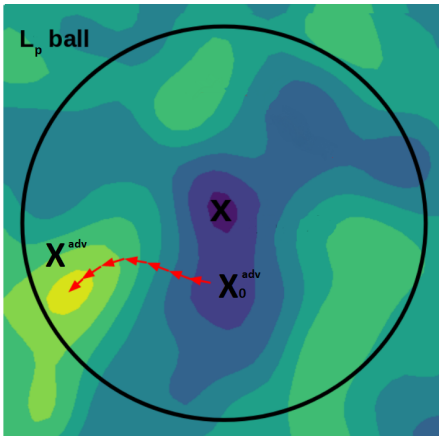

Figure 2. PGD after n iterations found the highest value for loss function. Figure is adopted from [12].

4.4 Generative adversarial networks

Method firstly proposed by Ian Goodfellow et al. in 2014 [9]. The main principal of Generative adversarial networks (GAN) is that two networks are fighting against each other. These networks are usually called discriminator D and generator G. G generates samples and D evaluates probability that sample comes from original data or is generated by G. Goal of G is to maximize error rate of D, which is, certainly, trying to have the best accuracy in classifying samples. This leads to *saddle point* problem and to treat it through the lens of *Game theory* the GAN's trying to find Nash equilibrium. Networks D and G are playing 2-player *minimax game* with value function $V(G,D)$:

$$\min_G \max_D V(G,D) = \mathbb{E}_{X\sim p_{data}(X)}[log(D(X))] + \mathbb{E}_{Z\sim p_Z(Z)}[log(1 - D(G(Z)))] \quad (6)$$

4.4.1 GAN for generating adversarial examples

In 2019 was proposed framework AdvGAN in [4] and took the first place in MNIST Challenge[3] in the black-box category. This is the only one *black-box* attack described and used in this paper. The purpose of 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 any target model. This is also a reason why is *AdvGAN* much

[2] projecting back to L^p ball means to move to the closet point inside the L^p ball

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

more effective than other adversarial attacks. When the generator is trained, we can generate perturbation for a sample using a 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 to experiment also with this black-box attack.

4.4.2 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). 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 the predicted sample and target class. The output of D is the adversarial loss [9]:

$$\begin{aligned} L_{GAN} = & \\ & \mathbb{E}_{X \sim p_{data}} log\, D(x) \\ & + \mathbb{E}_{X \sim p_{data}} log(1 - D(x + G(x))) \end{aligned} \tag{7}$$

The loss of target model is:

$$L_{adv} = \mathbb{E}_{X \sim p_{data}} l_f(x + G(x), t) \tag{8}$$

where t is target class, f is target model and l_f is loss function of target model. This loss is used for a targeted attack, an untargeted attack could be performed by maximizing the distance between prediction and the sample's label. 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 [13] (unlike referenced work we use L^∞ norm instead of L^2). Hinge loss is formally expressed as follows:

$$L_{hinge} = \mathbb{E}_{X \sim p_{data}} max(L^\infty(G(X)) - c, 0) \tag{9}$$

Finally, the generator is trained on the objective function

$$L = L_{adv} + \alpha * L_{GAN} + \beta * L_{hinge} \tag{10}$$

where α and β are experimentally chosen constants and represent the relative importance of loss functions.

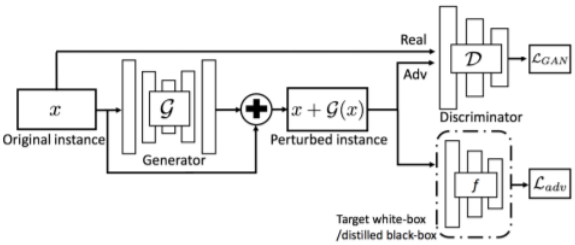

Figure 3. Architecture of AdvGAN. Figure is adopted from [4]

5. Defence

Our framework is designed to experiment with attacks and defences and it provides a full implementation of the method proposed by Madry et al. [3]. First of all, we need to present and explain a few observations that reasoned the correctness of methods in our framework.

5.1 Observations

Selection of parameters for our method is partially based on these observations and they are also important for building a robust model. We are providing adversarial training for any model, however, for best results user should beware these discoveries.

5.1.1 Capacity of network

Madry et al. in their work [3] show that just increasing the network's capacity helps increasing robustness against adversarial examples, while the network is trained only on natural samples. We can achieve only small improvement by using better architecture on natural data, but increasing capacity could improve the accuracy of a network on adversarial examples by more than 10%.

5.1.2 Label leaking

Observation by Kurakin et al. [8] that training on adversarial examples generated by single-step methods (especially FGSM) does not yield model robust against other attack strategies, because models trained on such samples use overfitting. We believe this is because transformations by single-step methods are simple and model can easily recognize them. The experiment supporting this statement is described in subsection 6.3.1.

5.2 Saddle point problem

Defence strategy implemented in this work is based on optimization problem called *saddle point* or *minimax* problem. A saddle point is a point on the graph of a function where the derivatives in orthogonal directions are all zero, however it is not a local extreme. We are finding such point to minimize maximization an error of our model using adversarial methods Formal expression of the task for our framework follows:

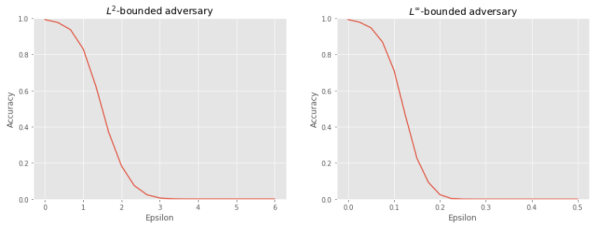

Figure 4. Comparison training convergence of L^2 and L^∞ norms. Figure is adopted from [12].

$$\underset{\theta}{argmin}\, E_{(x,y)\sim D}[\underset{\delta}{max}\, L(x+\delta, y, \theta)] \qquad (11)$$

where θ are model parameters, D is data distribution, L is loss function and δ is perturbation such that $\delta \in S$, where S is set of allowed perturbations.

5.3 Optimization against PGD samples

The method is introduced by Madry et al. [3], they also stated that the training model against PGD attacks will satisfy robustness against all methods using only first-order information.

5.3.1 Robustness wrt. *first-order attacks*

Every usage of *first-order methods* in this work will refer to *first-order methods* in **Optimization theory**. Generally, n-order methods is an algorithm that requires at least one n-th derivation/gradient. In this work are discussed only *first-order methods*.

It doesn't matter what is the starting point of PGD algorithm (we can select it randomly or set a starting point as an original sample) - the local maximum losses are almost the same in all cases. This suggests a view on the problem in which robustness against the PGD-generated samples yields robustness against all first-order samples. There is still a possibility of existence some local maximum with much higher value, but according to experiments [3] it is hard to find.

This assumption implies that we can build model robust against all first-order methods and the vast majority of optimization tasks in ML are solved using first-order, therefore the framework can yield model that is safe against state-of-the-art attacks.

6. Experiments

This section contains reasoning for using hyperparameters and experiments demonstrating the results of this work. Every experiment uses $\varepsilon = 0.3$ and we use only one network architecture[4] in another case it will be explicitly stated.

[4]Network architecture is defined here github.com/sio13/turtleNet/blob/master/target_model.py

Dataset	Natural	PGD	FGSM	BIM
MNIST	99.1%	6.9%	12.8%	7.4%
CIFAR10	70.6%	5.5%	8.1%	12.1%
CIFAR100	29.8%	1.2%	1.3%	2.2%

Table 1. Comparing three attacks on three models

6.1 Choosing the norm and value of ε

For the problem of adversarial examples are commonly used two norms - L^2 and L^∞. According to [12] is experimentally shown that using L^∞ it is possible to generate effective adversarial examples using 10 times smaller ε (see Figure 4). In the Figure 4 we can also see that it is useless to use greater ε than 0.3 (measured with the L^∞ norm) for generating adversarial examples, and this also the reason why we are using this value of ε is this work.

6.2 Attacking network

In this experiment, we demonstrate the vulnerability of neural networks by using three attack strategies. Target models trained on MNIST, CIFAR10 and CIFAR100 datasets achieve accuracy outstanding accuracy. By applying FGSM attack we can generate 10000 malign samples on which models achieve significantly lower accuracy and the generation process is really fast (for imagination it won't last more than 10 seconds on any laptop). On the other hand, the most successful attack was PGD, as we expected. Target models achieve an accuracy as little below 10% for all datasets. Generation process lasted about 10-times more than with FGSM. See the experiment results in Table 1.

Another experiment with adversarial attacks was more focused on MNIST trained model. For more detailed statistics see Figure 6. Notice, that the fastest attack is with method AdvGAN because the adversarial generator is already trained.

6.3 Comparison of target adversaries

This subsection contains experiments related to training against adversarial examples generated by some methods and comparison of its results.

6.3.1 Single-step method

The first approach to train model against adversarial examples was trying to train it using FGSM examples. This method seems to work well against FGSM examples and is also effective. Since it is a one-step method and it is much faster than iterative methods. Training with these samples and a suitable number of iterations yields model that achieves against FGSM accuracy more than 80%, but against other methods, it fails (see Figure 5). This behaviour is probably caused

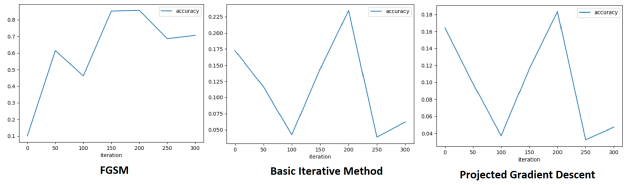

Figure 5. Comparison training accuracy of a model trained on FGSM examples against 3 adversarial methods.

attack	accuracy	loss	attack_time
MomentumIterativeMethod	0.09269999712705601	9.61143680114746	53.580769777297974
MadryEtAl	0.06849999725818601	12.184943614196778	53.068865060806274
BasicIterativeMethod	0.07029999792575 8	14.372136480712891	53.0767183303833
ProjectedGradientDescent	0.06840000301599501	12.217894227600098	54.56422805786133
FastGradientMethod	0.15090000629425002	2.641806679916381	5.482558012008667
AdvGAN	0.10120000529425102	7.848866679916381	2.569878012008667
None	0.9886000156402588	0.040568183930319995	0

Figure 6. Table showing damage of attacks to the naturally trained network.

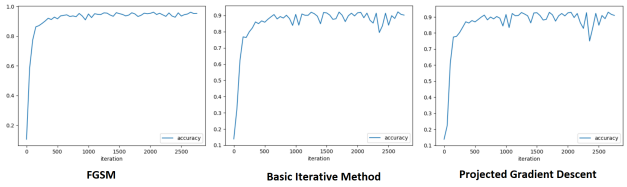

Figure 7. Comparison training accuracy of a model trained on PGD examples against 3 adversarial methods.

by overfitting to samples generated using a too simple method (see subsection 5.1.2).

Training against this method is just slightly better than no adversarial training 6. After about 200 iteration training with FGSM samples model achieves just less than 20%, which is not much more comparing to 6.5% achieved by naturally trained model.

6.3.2 PGD for training

Another and much more successful method for adversarial training is using PGD samples training. The method is much more resources consuming because it requires a high amount of iterations and every iteration requires the generation of PGD samples, but it converges to high accuracy. Using this method we can achieve accuracy more than 90% against attacks showed in Figure 7. Robustness was demonstrated also against other first-order methods, all experiments are visualized in the public repository[5].

7. Conclusions

Along with this work, we implemented a Python library called *turtleNet* with the implemented method for attacking and training a robust network.

[5] https://github.com/sio13/turtleNet/tree/master/result_pictures

This work also provided evidence about the vulnerability of neural networks to synthetically generated malignant samples. We also showed and implemented the solution for this issue as a part of *turtleNet* and experimentally proved that it relatively satisfies current requirements. Our implementation should be used for further experiments and for performing adversarial training on many other models. The main advantage of the framework is a possibility to make robust a model implemented using Keras or any other library. The only requirement is to convert a model into h5 format. We also provide an easy way to verify models robustness using several adversarial attacks.

Even we showed that we can face first-order adversaries, which are most common and easy to realize, there are still ways how to attack resistant model with high damage. It is almost clear that enemies will always improve their strategies and the only way how to keep this field safe is to constantly research new and better defence methods.

This research does not aim to change the world. It's main and greatest goal is to prevent changing world of machine learning and AI the way we don't want to.

7.0.1 Weaknesses

Our research assumes some constant value of ε, however, there exist samples with high distance from original data, but looks similar, therefore there is still a possibility to easily generate adversarial examples with higher ε, which look pretty similar to natural data and fool the mode this way.

7.0.2 Future work

The following research should investigate higher-order attacks and explore possibilities to fool our robust model. An idea behind *turtleNet* is to create software that unifies a large amount of attack and defence strategies for neural networks. This goal is not satisfied at all and it requires a lot of work in the future.

Acknowledgements

I would like to thank my supervisor RNDr. Milan Češka, Ph.D. for his help.

References

[1] Zeshan Kurd, Tim Kelly, and Jim Austin. Developing artificial neural networks for safety critical systems, 2006.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015. https://arxiv.org/pdf/1412.6572.pdf.

[3] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[4] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks, 2019.

[5] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning, 2017. https://arxiv.org/pdf/1602.02697.pdf.

[6] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014. https://arxiv.org/pdf/1312.6199.pdf.

[7] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks, 2016. https://arxiv.org/pdf/1610.06940.pdf.

[8] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale, 2017. https://arxiv.org/pdf/1611.01236.pdf.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, David Warde-Farley Bing Xu, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets, 2014. https://arxiv.org/pdf/1406.2661.pdf.

[10] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world, 2017. https://arxiv.org/pdf/1607.02533.pdf.

[11] Tianhang Zheng, Changyou Chen, and Kui Ren. Is pgd-adversarial training necessary? alternative training via a soft-quantization network with noisy-natural samples only, 2019. https://arxiv.org/abs/1810.05665.

[12] Oscar Knagg. Know your enemy, 2019. https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3.

[13] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017. https://arxiv.org/pdf/1608.04644.pdf.