KU LEUVEN

FACULTY OF
ENGINEERING SCIENCE

Distributed Adversarial Attacks

Sander Prenen

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Artificiële intelligentie

Promotoren:
Prof. dr. ir. W. Joosen
Dr. ir. D. Preuveneers

Assessoren:
Ir. W. Eetveel
W. Eetrest

Begeleiders:
V. Rimmer
I. Tsingenopoulos

Academiejaar 2021 – 2022

© Copyright KU Leuven

Without written permission of the thesis supervisors and the author it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to the Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 or by email `info@cs.kuleuven.be`.

A written permission of the thesis supervisors is also required to use the methods, products, schematics and programmes described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

Zonder voorafgaande schriftelijke toestemming van zowel de promotoren als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail `info@cs.kuleuven.be`.

Voorafgaande schriftelijke toestemming van de promotoren is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Preface

Sander Prenen

Contents

Preface **i**

Contents **ii**

List of Figures **ii**

List of Tables **iii**

1 Introduction **1**

2 Background **3**
- 2.1 Neural networks 3
- 2.2 Adversarial attacks 5
- 2.3 Particle swarm optimization 8

3 Related work **11**
- 3.1 Boundary attack 11
- 3.2 HopSkipJumpAttack 12
- 3.3 Stateful defense 14
- 3.4 PSO and distributed attacks 15

Bibliography **17**

List of Figures

2.1 Linear separability 4
2.2 Decision boundaries and decision regions 4
2.3 Activiation functions 4
2.4 Difference targeted and untargeted attack 6
2.5 Adversarial training 8
2.6 Particle swarm optimization 9

3.1 Intuition of the Boundary Attack 13
3.2 Intuition of the HopSkipJumpAttack 14

List of Tables

List of Abbreviations

AI Artificial Intelligence. 3

ANN Artificial Neural Network. 3, 5

API Application Programming Interface. 5

BA Boundary Attack. 11, 12, 13

BBA Biased Boundary Attack. 12, 13

CNN Convolutional Neural Network. 3

DDoS Distributed Denial of Service. 15

DkNN Deep k-Nearest Neighbors. 8

DNN Deep Neural Network. 3

EA Evolutionary Algorithm. 8, 15

FGSM Fast Gradient Sign Method. 5

HSJA HopSkipJumpAttack. 12, 13

PSO Particle Swarm Optimization. 8, 9, 15, 16

ReLU Rectified Linear Unit. 3, 4

RNN Recurrent Neural Network. 3

Chapter 1

Introduction

Chapter 2

Background

2.1 Neural networks

Ever since the invention of computer systems, it has always been a goal of scientists and engineers to create Artificial Intelligence (AI). Current state of the art approaches are mimicking the human brain, more specifically the neurons inside the brain. Already in the fifties, Rosenblatt introduced his perceptron [1]. The perceptron is a single neuron able to learn linearly separable patterns. It does so by finding a hyperplane that separates the two classes. This hyperplane is called the decision surface or decision boundary and the perceptron itself is called a classifier. Geometric regions separated by a decision boundary are called decision regions. The concept of linear separability is explained in Figure 2.1 in two dimensions. In Figure 2.2, the decision boundaries and decision regions are explained visually.

Unfortunately not all patterns are linearly separable. To overcome this problem, the neurons can be layered, creating an Artificial Neural Network (ANN) in the process. Layering neurons sequentially is essentially a linear combination of neurons. This in itself does not create non-linear decision surfaces. Non-linear activation functions are added for the ANN to be able to learn more complex decision boundaries. Some commonly used activation functions are Rectified Linear Unit (ReLU) [2], Heaviside step function and softmax (or sigmoid when used on scalars). In Figure 2.3 the plots of the activation functions can be found.

The neurons can be combined in different ways to create different ANN architectures. Each architecture has its own strengths and weaknesses. Convolutional Neural Networks (CNNs) excel in classifying visual data [3, 4], whilst Recurrent Neural Networks (RNNs) are widely used when there exist dependencies inside the data, such as in speech recognition [5, 6] or time series prediction [7]. More recent research focuses on Deep Neural Networks (DNNs), due to the ever increasing computational power available. DNN approaches are able to compare to and even surpass human performance [8, 9].

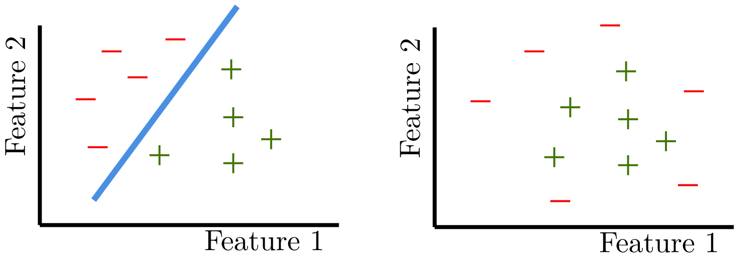

Figure 2.1: Linearly separable classes on the left and non-linearly separable classes on the right. Two classes are linearly separable if there exists a hyperplane for which all examples of one class are on the same side of this hyperplane, whilst all examples of the other class are on the other side of the hyperplane. In two dimensions, the hyperplane is a straight line.

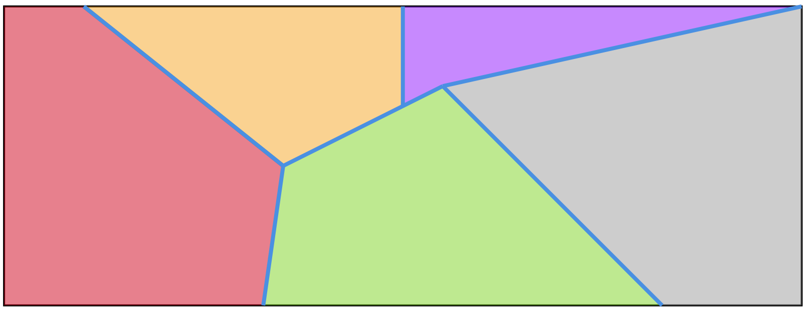

Figure 2.2: Decision boundaries (blue lines) separate different decision regions (colored regions).

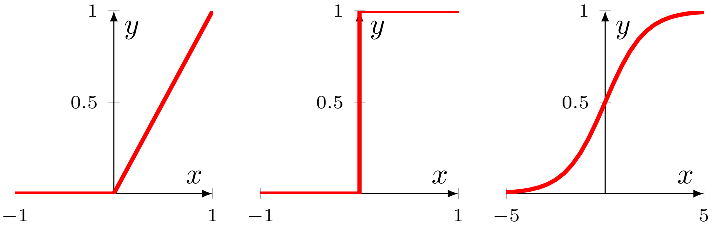

Figure 2.3: Plots of different activation functions. From left to right: ReLU, Heaviside step and sigmoid.

2.2 Adversarial attacks

The expressiveness of ANNs is a double-edged sword. It is the cause for the near-human performance on some tasks, but also for counter-intuitive properties. As studied by Szegedy et al [10], one of these properties is the presence of discontinuous decision boundaries. This might cause seemingly identically images to be classified differently. They first defined adversarial examples as *imperceptibly small perturbations to a correctly classified input image, so that it is no longer classified correctly* [10]. This property of ANNs might not seem important at first glance, but it can be quite worrisome from a security point-of-view. Malicious users could craft images to bypass face recognition software [11] or attack the camera of a self-driving car to misclassify traffic signs [12]. Other fields where adversarial examples are of interest include malware detection [13], natural language processing [14] and industrial control systems [15]. Adversarial attacks are algorithms used to craft such adversarial examples.

Most research on adversarial attacks is done using images. Researchers have the most freedom in this domain, since a slightly altered image is still an image with roughly the same contents. Slightly modifying an industrial control system however, might break the entire way the system works. Research in other domains is mostly conducted by altering existing image algorithms to the specific use case. For this reason this works only focuses on adversarial attacks on images.

TODO: Threat model!

2.2.1 Adversarial attacks terminology

Adversarial attacks are generally divided in two categories, white box attacks and black box attacks. In a white box attack, the attacker has complete knowledge of the classifier under attack. This knowledge consists of the architecture, parameters and thus their gradients and all output of the classifier. Examples of white box attacks are the Fast Gradient Sign Method (FGSM) [16] or the Carlini & Wagner attack [17].

In black box attacks, the only thing the attacker has access to is the output of the model. Depending on the literature, this output consists of class labels only (decision-based attack) or class labels and the corresponding confidence scores (score-based attacks). Black box attacks are more relevant in real-life scenarios, since most attacks are performed on a third-party Application Programming Interface (API). These APIs generally do not reveal the underlying model.

Transfer attacks [18] try to overcome this hurdle by creating a surrogate model. This is an undefended model similar to the model under attack. This idea is based on the observation that adversarial perturbations often are transferable to other models [16]. Attacks can leverage information (such as gradients) from the surrogate model

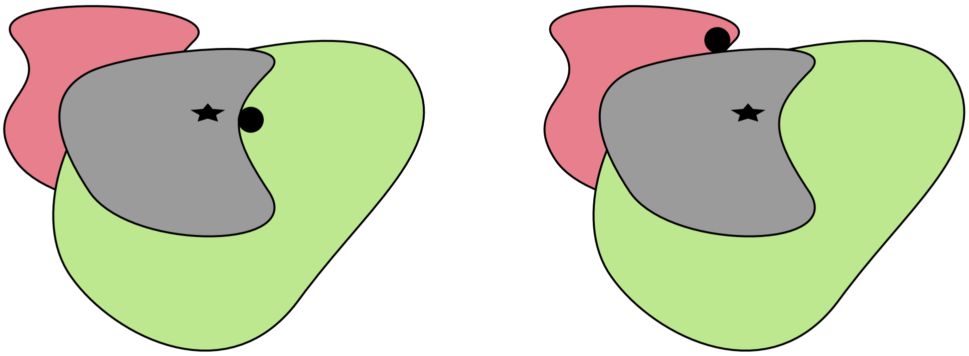

Figure 2.4: Different decision regions are shown in different colors. An adversarial example is being created starting from the source image (black star). On the left an untargeted attack is performed. The adversarial example is the image closest to the source image, that is classified differently (black circle). On the right a targeted attack is performed with the red decision region being the target class. The adversarial example is the image closest to the source image that is in the red decision region.

to breach the black box model. Due to the transferability of adversarial examples, it is also possible to perform so called zero-query attacks. Zero-query attacks are performed entirely on the surrogate model and the resulting adversarial example is forwarded to the black box model.

Both white box and black box attacks can be divided into targeted and untargeted attacks depending on their goal. In a targeted attack, the goal of the attacker is to create an adversarial example with a specific target class. In an untargeted attack the target class can be any class. Untargeted variants of attacks generally enjoy much more freedom and are therefore able to craft adversarial examples that are closer to the original. Figure 2.4 visually explains the difference between the two types.

What does it mean for images to be close to each other? This is easy to visualize in two dimensions as in Figure 2.4, but in higher dimensions, this is more difficult. Images reside in d-dimensional space, where d is the amount of pixels of the image. Two commonly used distances in higher dimensions are the L_2-distance and the L_∞-distance. They are defined as follows:

$$
\begin{aligned}
L_2(X,Y) &= \sqrt{\sum_{i=1}^{d} |x_i - y_i|^2} \\
L_\infty(X,Y) &= \lim_{p\to\infty} \left(\sum_{i=1}^{d} |x_i - y_i|^p\right)^{1/p} \\
&= \max(|x_1 - y_1|, |x_2 - y_2|, \dots, |x_d - y_d|)
\end{aligned}
$$

In both distances X and Y represent the images and $(x_1, x_2, \dots, x_i, \dots, x_d)$ and $(y_1, y_2, \dots, y_i, \dots, y_d)$ are the pixel values of X and Y respectively. The L_2-distance is also known as the Euclidean distance, which is a generalization of the Pythagorean theorem in more than two dimensions. It takes the pairwise distances between all pixels into account. The L_∞-distance is also called the Chebyshev distance. This distance only depends on the maximal pairwise distance between the two images. By minimizing the L_∞-distance, the maximal pixelwise difference in minimized [19].

2.2.2 Adversarial defenses

The existence of adversarial attacks naturally gave rise to adversarial defenses. These defenses can be categorized based on their objective. They can be either proactive or reactive. The goal of a proactive defense consists of making the models under attack more robust, while reactive defenses aim to identify attacks before they reach the model [20]. Different defensive countermeasures can be taken in each category. The remainder of this section will discuss the some commonly used techniques [21].

Gradient masking techniques hinder optimization-based attacks by having gradients "*that are not useful*" [22]. They are also sometimes referred to as obfuscated gradients [23]. Three types of obfuscated gradients can be identified. Shattered gradients introduce incorrect or non-existent gradients. Stochastic gradients are caused by random effects in the defense and exploding or vanishing gradients are primarily caused by chaining neural network evaluations. Besides intentionally introducing gradient masking in neural networks, they can also be introduced unintentionally due to the design of the network.

Defensive distillation [24] is a technique that can be classified as gradient masking. The goal of defensive distillation is to smooth the gradients of the model, making it more resilient to small input perturbations. The distillation procedure is done in two steps. First the model is trained on the original data and labels. This step produces a probabilistic output for each input due to the softmax activation function. Then the network is retrained using the original data and the probabilistic outputs as labels. The probabilistic labels contain additional knowledge that can be exploited to the increase generalizabilty of the model.

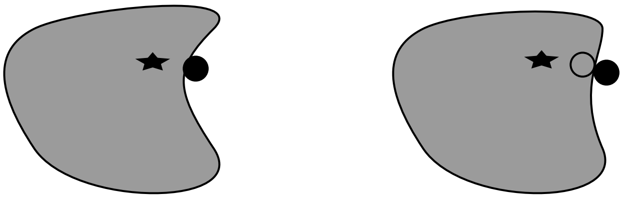

Figure 2.5: Decision boundaries before (left) and after (right) adversarial training. Before adversarial training, an adversarial example can be created (black circle). This example is added to the training set and the network is retrained. A new decision surface is created in the process. This surface classifies the previous adversarial example correctly (empty circle), but there is a new opportunity for an adversarial attack (black circle).

Adversarial training [16] techniques inject adversarial examples in the training dataset and retrain the model in order to create a more resilient model. This is essentially a brute force method to correctly classify some adversarial examples. However this new model is still susceptible to new adversarial attacks, since the decision boundary has only been moved slightly. This can be seen in Figure 2.5.

Preprocessing techniques work on the inputs of a model. Different preprocessing techniques, such as denoising [25], dimensionality reduction [26] and image transformations [27] can be used to defend the model under attack. The goal of all techniques boils down to giving the attacker less control over the exact input that is being fed to the model.

Some defenses rely on **proximity measurements** between the input and the model. An example of this countermeasure is Deep k-Nearest Neighbors (DkNN) by Papernot and McDaniel [28]. DkNN computes support for a decision from a network based on a nearest neighbors search in the training data. Another example is region-based classification [29], where a prediction is made for a given input based on the proximity of training examples.

2.3 Particle swarm optimization

Particle Swarm Optimization (PSO) [30] is an optimization framework part of the Evolutionary Algorithms (EAs) family. In EAs, populations of candidate solutions evolve based on mechanisms inspired by the field op biology, such as ant colonies [31], mutation and recombination [32]. The mechanism that inspired PSO is the flocking of birds. The framework has been applied to numerous problems such as routing problems [33, 34], diagnosing diseases from imaging [35] and calculating heat transfer coefficients [36].

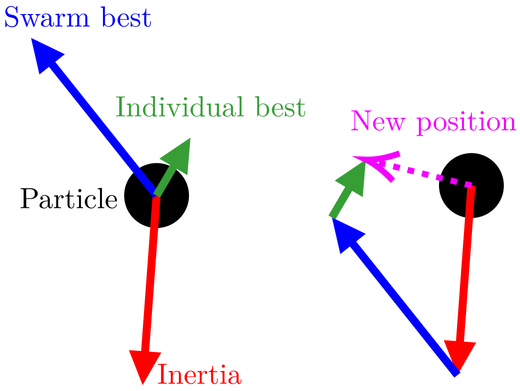

Figure 2.6: Particles move based on a step towards their best known position, the swarm's best known position and a step in the direction of the movement of the previous iteration. The different steps are combined to determine the new position of the particle.

In PSO, different particles move through the search space based on a set of rules. Each position in the search space has a given fitness value. This value determines how 'fit' or good a certain position is with respect to the goal of the optimization problem. Every iteration, all particles take a step towards their personal best position, towards the group or swarm's best position and a step in the current direction (inertia). The different step sizes can be weighted depending on the problem at hand. In Figure 2.6, the steps are graphically represented for a single particle.

Chapter 3

Related work

3.1 Boundary attack

Boundary Attack (BA) [37] is a decision-based adversarial attack. The basic intuition of BA differs from traditional adversarial attacks. Unlike these traditional adversarial attacks, where the original image is moved through search space in order to become adversarial, BA starts from an input that is already adversarial. This input is then moved closer to the original image, while staying adversarial.

The attack has to be initialized with an already adversarial input. Two different approaches can be taken depending on the attack setting. In the untargeted case, the input can be sampled from a maximum entropy distribution given the valid domain of this input. Samples that are not adversarial are rejected. In the case of a targeted attack, the input is a sample from the dataset that is classified as the target class by the model under attack.

BA iteratively updates the adversarial image by performing a step orthogonal to the original image and a step towards this image. In iteration k, a perturbation η_k is sampled from a Gaussian distribution. This perturbation is rescaled and added to the adversarial image. From this new position in search space, the step towards the original image is taken. This way the path of the attack follows the decision boundary, hence the name of the attack. The intuition of the BA is shown in Figure 3.1. The attack can only follow the boundary if the adversarial image is already near the boundary. The starting image is projected onto the boundary using binary search to ensure that the adversarial image is in the vicinity of the boundary.

The step sizes are adjusted according to local geometry of the boundary. The orthogonal step size δ is adjusted so that approximately half of the orthogonal perturbations is still adversarial. This approach is based on trust region methods [38]. The step size towards the original image ϵ is adjusted using the same principle, but here a user specified threshold is used. The decision boundary tends to become flatter, the closer to the original image the attack gets. Therefore the algorithm converges when

ϵ converges to zero.

Biased Boundary Attack (BBA) [39] (previously known as *Boundary Attack++*) improves the original BA in three different ways. All three improvements will be discussed in order of the strength of their effect. The first improvement is a biased sampling technique. The key idea behind this that most previous attacks yield adversarial examples with high frequencies in the image. By sampling the perturbations in the first step of the BA from a low frequency distribution, the frequency of the created adversarial example will be lowered as well. BBA does this by sampling from a Perlin noise [40] distribution. Lower frequency images yield more natural results and can more easily bypass simple preprocessing defense schemes.

The second improvement is to use a masking. The original BA applies a perturbation to the images as a whole. Every pixel will be perturbed with the same magnitude. This magnitude can be altered on a per-pixel basis when using a mask. Pixels that are farther away from the target image will receive a larger perturbation than pixels that are already close to the corresponding pixel in the target. This technique improves efficiency since the search space is significantly reduced. It is also possible to engineer masks for specific examples in order to incorporate other knowledge in the attack.

The final improvement is based on the idea of transfer attacks. A surrogate model is trained and will be used to calculate adversarial gradients. These gradients will then be used to bias the sampling direction for the orthogonal step. If the surrogate model does not closely resemble the defender, then the gradients will only hamper the speed of convergence of the attack instead of completely failing the attack.

3.2 HopSkipJumpAttack

HopSkipJumpAttack (HSJA) [41], like BA, is a decision-based adversarial attack that starts from an adversarial input. The initial input is obtained in an identical manner as in BA. HSJA is an iterative algorithm that consists of three steps.

The first step is a projection onto the decision boundary of the model under attack. This projection is carried out using a binary search. The second step is to estimate the direction of the gradient at the boundary. Different directions are sampled from a uniform distribution over a d-dimensional sphere, where d is the input dimension. This random direction is added to the boundary point, generating a new query for the model. The results of these queries are combined to a gradient estimation $\widetilde{\nabla S}$ using the Monte Carlo estimate of equation 3.1. In this equation u_b are the random directions and x_t is the boundary position. B is the number of random directions that needs to be sampled. This number increases based on the current iteration of the attack to reduce the variance of the estimate. The function ϕ_{x^*} returns 1 if the new position is adversarial and -1 if it is not adversarial. δ is a positive parameter

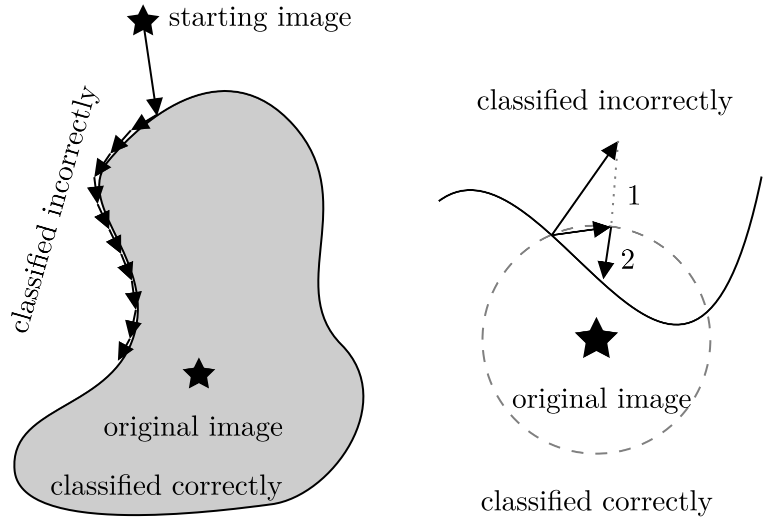

Figure 3.1: Intuition behind the Boundary Attack. On the left the path of the attack is shown. The first step is a projection onto the boundary, afterwards it follows the decision boundary of the class of the original image. Each arrow represents one iteration of the attack. On the right, the two different steps of each iteration can be seen. In the first step, a random direction is sampled and projected onto a sphere around the original image. The second step is to take a step towards the original image from this new position. Image inspired by [37].

determining the size of the d-dimensional sphere.

$$\widetilde{\nabla S}(x_t, \delta) := \frac{1}{B} \sum_{b=1}^{B} \phi_{x^*}(x_t + \delta u_b) u_b \tag{3.1}$$

Once the gradient has been estimated, the third and final operation is to take a step along this gradient. The step size is determined using a geometric progression scheme. These steps are iteratively repeated until the pre-set stopping criterion is met. Figure 3.2 represents the intuition behind HSJA in a graphical manner.

HSJA eclipses BA and BBA both on median distance against queries and attack success rates using a limited amount of queries. The untargeted version of HSJA is able to compete with white box attacks on the ImageNet dataset [42]. It also performs similar or superior to white box attacks such as the C&W attack [17] when evaluated against defensive mechanisms such as defensive distillation [24], region-based classification [29] and adversarial training [16].

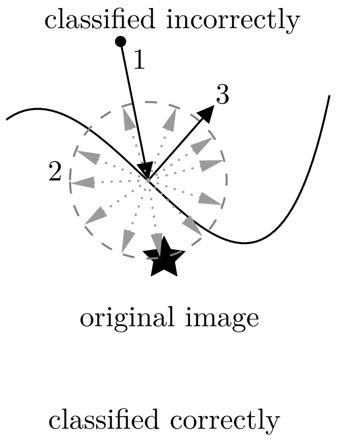

Figure 3.2: Intuition behind the HopSkipJumpAttack. Each iteration consists of three steps. The first step is a projection onto the boundary. The second step is the estimation of the gradient at this point. This is done by sampling directions from a uniform distribution and querying the model under attack from this new position (grey arrows). The results are combined via the Monte Carlo estimate. The third and final step is to take a step along the estimated gradient. Image inspired by [41].

3.3 Stateful defense

The defensive schemes discussed in section 2.2.2 all operate on the query level. They try to detect and flag possible attacks based on a single query without taking other context into account. The stateful detection mechanism by Chen, Carlini and Wagner [43] is different in this aspect. As the name suggests, it holds state of previously submitted queries. It is similar to the defenses that use proximity measurements, but the measurement is between queries instead of between the query and training data.

All queries submitted to the model equipped with a stateful detection mechanism are stored in a history buffer. Each user of the model has a distinct history buffer, where its queries are stored. These buffers can be bounded by time or number of queries depending on the resources available and the use case of the model. Each time a query is submitted to the model, the average distance to its k nearest neighbors is calculated and if this distance is lower than a certain threshold then the user gets flagged by the mechanism. Appropriate actions such as banning the account can be taken.

The distance metric is not calculated in input space. Each query is encoded by a deep similarity encoder [44] to an encoded space, typically of a lower dimension. In this encoded space, images which represent perceptually similar objects are clustered together. The advantage of the encoded space is twofold. Firstly, the dimension of the encoded space is smaller than the dimension of the input space. Therefore less space is needed to store the history buffers. Secondly, simpler distance metrics such as L_2-distance in input space can easily be evaded by an attacker. For example the L_2-distance can be significantly increased by simply rotating the input image.

The parameter k, the number of neighbors to consider is picked as follows. As the training data of the model consists of only benign queries, no attacks should be flagged when feeding the stateful detection mechanism with this data. To allow for some more leniency, a false positive rate of 0.1% is still acceptable. For each value of k, a different threshold will be required to maintain the selected false positive rate. Larger values have the benefit of larger thresholds causing the defense to be more resilient, since attackers images need to be more diverse. But k is also the number of queries needed before an attack can be flagged. Therefore too large values for k are disadvantageous. Smaller values also reduce computational cost. Chen, Carlini and Wagner set the value of k to 50 for the CIFAR-10 dataset [45], since the thresholds increased sharply up to this value. Other datasets require different values for k.

3.4 PSO and distributed attacks

There have been several attempts to craft adversarial examples using an EA. Most previous attempts try to reduce the number of queries needed to create a successful adversarial example [46, 47, 48, 49]. While other approaches are more focused on using EAs to evade detection mechanisms [50].

GenAttack [46] and the similar efficient attack by Dong et al. [47] use genetic algorithms in order to minimize the number of queries to the model. Both algorithms reduce the dimension of the search space to improve the efficiency attack. Once a promising perturbation is found in this lower dimensional space, it is upscaled using a bilinear transformation. By reducing the search space, the number of individuals in the genetic algorithm can be lowered, which in turn lowers the total amount of queries. GenAttack also uses annealing schemes to adaptively scale the parameters of the algorithm. This allows it to escape local optima and improve the adversarial example further.

AdversarialPSO [48] and the similar attack from [49] use PSO as optimization routine on images and audio fragments respectively. Each particle represents a possible adversarial example. Both attacks use the standard rules of PSO as specified by [30] improved with a linearly decaying inertia weight [51]. The former attack also uses a constriction factor to avoid premature convergence [52]. While the latter solves this problem by generating new particles using a genetic algorithm when premature convergence is detected.

The idea behind the multi-group PSO attack [50] is to use PSO to evade detection by the stateful detection mechanism described in section 3.3. The intuition behind it is inspired by the Distributed Denial of Service (DDoS) attack [53]. The swarm is split into multiple smaller groups and each group is placed on a single node. The groups perform the standard PSO algorithm. The best position of the swarm as a whole is communicated using a dedicated server. Each groups submits its queries from its own node, tricking the defensive mechanism into thinking that multiple users are submitting queries. The mechanism will create a history buffer for each group as opposed to a single buffer for the whole swarm. This ultimately results in less detections.

Bibliography

[1] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. 65(6):386–408.

[2] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[3] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, and N.I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 177:232–243, 2020.

[4] Steve Lawrence, C. Lee Giles, Ah Chung Tsoi, and Andrew D. Back. Face recognition: A convolutional neural network approach. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 8(1):98–113, 1997.

[5] Xiangang Li and Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. *CoRR*, abs/1410.4281, 2014.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[7] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *CoRR*, abs/1909.00590, 2019.

[8] Jon Russell. Google's alphago ai wins three-match series against the world's best go player. `https://techcrunch.com/2017/05/24/alphago-beats-planets-best-human-go-player-ke-jie/amp/?guccounter=1`, 05 2017. Accessed: 2021-12-08.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[10] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

[11] Fatemeh Vakhshiteh, Ahmad Nickabadi, and Raghavendra Ramachandra. Adversarial attacks against face recognition: A comprehensive study, 2021.

[12] Abhiram Gnanasambandam, Alex M. Sherman, and Stanley H. Chan. Optical adversarial attack, 2021.

[13] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection, 2019.

[14] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text, 2018.

[15] Giulio Zizzo, Chris Hankin, Sergio Maffeis, and Kevin Jones. Invited: Adversarial machine learning beyond the image domain. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–4, 2019.

[16] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017.

[18] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2018.

[19] Distance. Distance. `https://en.wikipedia.org/wiki/Distance`, 11 2021. Accessed: 2021-12-09.

[20] Gabriel Resende Machado, Eugênio Silva, and Ronaldo Ribeiro Goldschmidt. Adversarial machine learning in image classification: A survey toward the defender's perspective. *ACM Comput. Surv.*, 55(1), nov 2021.

[21] Gabriel Resende Machado, Eugênio Silva, and Ronaldo Ribeiro Goldschmidt. Adversarial machine learning in image classification: A survey towards the defender's perspective. *CoRR*, abs/2009.03728, 2020.

[22] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 506–519, New York, NY, USA, 2017. Association for Computing Machinery.

[23] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *CoRR*, abs/1802.00420, 2018.

[24] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1511.04508, 2015.

[25] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples, 2015.

[26] Dan Hendrycks and Kevin Gimpel. Visible progress on adversarial images and a new saliency map. *CoRR*, abs/1608.00530, 2016.

[27] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. *CoRR*, abs/1711.00117, 2017.

[28] Nicolas Papernot and Patrick D. McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *CoRR*, abs/1803.04765, 2018.

[29] Xiaoyu Cao and Neil Zhenqiang Gong. Mitigating evasion attacks to deep neural networks via region-based classification. *CoRR*, abs/1709.05583, 2017.

[30] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[31] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. ieee trans syst man cybernetics - part b. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 26:29–41, 02 1996.

[32] John H. Holland. *Genetic Algorithms and Adaptation*, pages 317–333. Springer US, Boston, MA, 1984.

[33] Rami Abousleiman and Osamah Rawashdeh. Electric vehicle modelling and energy-efficient routing using particle swarm optimisation. *IET Intelligent Transport Systems*, 10(2):65–72, 2016.

[34] Tadashi Yamada and Zukhruf Febri. Freight transport network design using particle swarm optimisation in supply chain–transport supernetwork equilibrium. *Transportation Research Part E: Logistics and Transportation Review*, 75:164–187, 2015.

[35] Bruno Almeida and Victor Coppo Leite. *Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems.* 12 2019.

[36] Romeela Mohee and Ackmez Mudhoo. Analysis of the physical properties of an in-vessel composting matrix. *Powder Technology*, 155(1):92–99, 2005.

[37] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. *arXiv:1712.04248 [cs, stat]*, February 2018. arXiv: 1712.04248.

[38] Ye Wenhe. Trust-region methods. https://optimization.mccormick.northwestern.edu/index.php/Trust-region_methods, 06 2014. Accessed: 2021-12-09.

[39] Thomas Brunner, Frederik Diehl, Michael Truong Le, and Alois Knoll. Guessing Smart: Biased Sampling for Efficient Black-Box Adversarial Attacks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4957–4965, October 2019. arXiv: 1812.09803.

[40] Ken Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, page 287–296, New York, NY, USA, 1985. Association for Computing Machinery.

[41] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. *arXiv:1904.02144 [cs, math, stat]*, April 2020. arXiv: 1904.02144.

[42] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[43] Steven Chen, Nicholas Carlini, and David Wagner. Stateful Detection of Black-Box Adversarial Attacks. *arXiv:1907.05587 [cs]*, July 2019. arXiv: 1907.05587.

[44] Sean Bell and Kavita Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Trans. Graph.*, 34(4), jul 2015.

[45] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[46] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani Srivastava. GenAttack: Practical Black-box Attacks with Gradient-Free Optimization. *arXiv:1805.11090 [cs]*, June 2019. arXiv: 1805.11090.

[47] Yinpeng Dong, Hang Su, Baoyuan Wu, Zhifeng Li, Wei Liu, Tong Zhang, and Jun Zhu. Efficient decision-based black-box adversarial attacks on face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7714–7722, 2019.

[48] Rayan Mosli, Matthew Wright, Bo Yuan, and Yin Pan. They might not be giants: crafting black-box adversarial examples with fewer queries using particle swarm optimization. *arXiv preprint arXiv:1909.07490*, 2019.

[49] Hyunjun Mun, Sunggwan Seo, Baehoon Son, and Joobeom Yun. Black-box audio adversarial attack using particle swarm optimization. *IEEE Access*, 10:23532–23544, 2022.

[50] Naufal Suryanto, Hyoeun Kang, Yongsu Kim, Youngyeo Yun, Harashta Tatimma Larasati, and Howon Kim. A distributed black-box adversarial attack based on multi-group particle swarm optimization. *Sensors*, 20(24), 2020.

[51] Y. Shi and R.C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950 Vol. 3, 1999.

[52] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[53] Kaspersky. What is a ddos attack? - ddos meaning. `www.usa.kaspersky.com/resource-center/threats/ddos-attacks`, 01 2021. Accessed: 2022-03-18.