

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Using Double Oracle Algorithm for Classification of Adversarial Actions

Prokop Šilhavý

Supervisor: Mgr. Branislav Božanský, Ph.D.

Field of study: Open Informatics

Subfield: Artificial Intelligence

May 2019

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šilhavý** Jméno: **Prokop** Osobní číslo: **434728**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačů**
Studijní program: **Otevřená informatika**
Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Využití algoritmu inkrementálního generování strategií pro klasifikaci akcí útočníka

Název diplomové práce anglicky:

Using Double Oracle Algorithm for Classification of Adversarial Actions

Pokyny pro vypracování:

Classifiers are used in security domains where the actions of an adversary are being classified as malicious or benign. This interaction can be modeled as a game; the strategy of one player corresponds to setting parameters of a classifier, the strategy of the opponent is to choose such an input that causes misclassification. These games can be solved to a bounded error using a double oracle method that incrementally builds a discrete version of this continuous game. The goal of the student is to 1. Implement a flexible software framework for using double oracle for an arbitrary classification problem with 2 classes, 2. Analyze the performance of double oracle depending on the number of features, the structure of benign data, the shape of the utility function of the adversary, and the used classifiers and the computation time spent on finding new best responses, 3. Identify for which classes of problems the double oracle algorithm can be used to find robust classification strategies.

Seznam doporučené literatury:

- [1] McMahan, H. B., Gordon, G. J., & Blum, A. (2003). Planning in the presence of cost functions controlled by an adversary. In Proceedings of the 20th International Conference on Machine Learning (ICML-03) (pp. 536-543).
- [2] Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., ... & Graepel, T. (2017). A unified game-theoretic approach to multiagent reinforcement learning. In Advances in Neural Information Processing Systems (pp. 4190-4203).
- [3] M. Brückner, C. Kanzow, and T. Scheffer. Static prediction games for adversarial learning problems. Journal of Machine Learning Research, 13(Sep):2617–2654, 2012.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Mgr. Branislav Bošanský, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **14.02.2019**

Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

Mgr. Branislav Bošanský, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Ing. Pavel Ripka, CSc.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

I would like to thank my supervisor, Mgr. Branislav Bošanský, Ph.D., for his patient guidance, helpful advises, and constructive criticism.

Furthermore, I would like to thank my family for their love and support during my study, and especially my sister Terezie for valuable help.

Finally, my thanks belong also to VO MetaCentrum, which provides distributed computing infrastructure, and which enables us to run all the experiments.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

Prague, May 22, 2019

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze 22. května 2019

Abstract

This thesis examines the usability of Double-Oracle algorithm for finding a Nash equilibrium in infinite games. Especially, it focuses on finding a robust solution for classification of adversarial action.

At first, we have formalized an adversarial classification problem as an almost zero-sum game with hard false-positive constraint in expectation. For this representation, we have found an algorithm, which gives us the exact value of the game.

Double Oracle applied in this game consists of three parts: slightly modified LP for solving the restricted game, general optimization for finding the attacker's best response, and a classifier for an approximation of the defender's best response.

We have created a framework for using DO for classification of adversarial actions, and we have evaluated it on predefined domains with various structures and a various number of dimensions. The experiments have been performed with three classifier types: decision tree, SVM, and neural network. The experimental results have shown that the algorithm converges, but the computation time grows fast with the number of dimensions.

Keywords: Double Oracle, adversarial machine learning, infinite games

Supervisor: Mgr. Branislav Bošanský, Ph.D.

Abstract

Diplomová práce se zabývá použitím algoritmu inkrementálního generování strategií v nekonečných hrách. Konkrétně se zaměřuje na jeho využití při klasifikaci akcí útočníka.

Nejprve jsme si formalizovali problém adversariální klasifikace jako hru se striktním omezením na chybu prvního typu v prostoru smířených strategií, která je téměř s nulovým součtem. K této reprezentaci jsme vytvořili algoritmus, který nám přesně určí hodnotu hry.

Algoritmus inkrementálního generování strategií se v tomto případě skládá ze tří částí: z lehce upraveného LP na řešení omezené hry, z obecné optimalizační funkce pro nalezení optimální reakce útočníka a z klasifikátoru, který přibližně hledá optimální reakci obránce.

Vytvořili jsme framework používající algoritmus inkrementálního generování strategií pro řešení problému klasifikace akcí útočníka a otestovali jsme ho na doménách s různorodou strukturou a s různě dimenzionálním prostorem akcí útočníka. Experimenty využívaly tři různé klasifikátory: rozhodovací stromy, SVM a neuronové sítě. Výsledky ukázaly, že algoritmus konverguje, ale jeho časová náročnost rapidně roste s počtem dimenzí prostoru útočnickových akcí.

Keywords: algoritmus inkrementálního generování strategií, adversariální klasifikace, nekonečné hry

Title translation: Využití algoritmu inkrementálního generování strategií pro klasifikaci akcí útočníka

Contents

| | | | |
|---|-----------|--|-----------|
| 1 Introduction | 1 | 6.3 Weights of Benign Points for Classifier Training | 37 |
| 1.1 Related Work | 1 | 6.4 Decision Tree | 38 |
| 1.2 Outline and Contributions | 2 | 6.4.1 Unlimited Decision Tree | 42 |
| 2 Introduction to Game Theory | 5 | 6.5 Support Vector Machine | 43 |
| 2.1 Normal-Form Games | 5 | 6.6 Neural Network | 46 |
| 2.2 Infinite Games | 7 | 6.7 Final Observations | 48 |
| 3 Solution Concepts | 11 | 7 Conclusions | 51 |
| 3.1 Nash Equilibrium | 11 | 7.1 Future Work | 52 |
| 3.2 Finding of a NE in a Zero-Sum NFG | 13 | A Bibliography | 55 |
| 3.3 Finding of a NE in a General-Sum NFG | 14 | B Experiments on Discretization Algorithm | 61 |
| 3.4 Finding of a NE in Infinite Games | 14 | C Framework Source Code | 63 |
| 3.5 Double-Oracle Algorithm | 15 | D CD Content | 67 |
| 3.6 Stackelberg Equilibrium | 16 | | |
| 4 Adversarial Classification as a Game | 17 | | |
| 4.1 General-Sum Game | 18 | | |
| 4.2 Finding a NE by Discretization | 19 | | |
| 4.3 Finding a NE by Double Oracle | 20 | | |
| 4.4 Zero-Sum Game | 20 | | |
| 4.4.1 Hard Constraint in Classification | 21 | | |
| 4.4.2 Hard Constraint in Expectation | 21 | | |
| 4.4.3 Soft Constraint in Expectation | 23 | | |
| 4.5 Specialized Discretization for Game with Hard Constraint in Expectation | 23 | | |
| 5 Framework Details and Setting of Experiments | 25 | | |
| 5.1 Datasets | 25 | | |
| 5.2 Specialized Discretization | 27 | | |
| 5.3 Double Oracle Setup | 27 | | |
| 5.3.1 Attacker's Best Response | 28 | | |
| 5.3.2 Defender's Best Response | 29 | | |
| 5.4 Implementation Details | 30 | | |
| 6 Experiments | 33 | | |
| 6.1 Alternating or Simultaneous BR Computation | 33 | | |
| 6.2 Comparison of Optimization Algorithms | 36 | | |
| 6.2.1 Insertion of Benign Points to the Optimization | 37 | | |

Figures

| | |
|--|---|
| <p>1.1 An example of the confusion of a classifier by addition of adversarial noise to the original image [1] 1</p> <p>2.1 A payoff matrix for Rock, Paper, Scissors game 6</p> <p>2.2 An example of a polynomial game 8</p> <p>2.3 An example of a non-separable game 10</p> <p>3.1 A payoff matrix for the Prisoner’s dilemma game [2] 12</p> <p>3.2 Schematic of the Double-Oracle algorithm. [3] 15</p> <p>4.1 An example of an adversarial classification game. The utility function is linear until the diagonal, where the defender’s threshold is. Behind the diagonal there is utility 0. Vertical lines stand for the discontinuity. 18</p> <p>4.2 An example of an adversarial classification general-sum game. Black lines correspond to the benign points, and vertical lines stand for the discontinuity. 18</p> <p>4.3 An example of a zero-sum adversarial classification game with a restricted false-positive rate of the classifiers 21</p> <p>5.1 The functions displayed for dimensions 1 and 2 26</p> <p>5.2 The datasets displayed for dimensions 1 and 2 27</p> <p>5.3 Schema of the used neural network 31</p> <p>6.1 A few iterations of the run with simultaneous BR computation 34</p> <p>6.2 A few iterations of the run with alternating BR computation. 35</p> <p>6.3 Time duration of the optimization algorithms 36</p> <p>6.4 Time duration of the BR algorithm, depending on the dimension of the attacker’s space . 40</p> | <p>6.5 Convergence of DO with the three-dimensional linear utility and DT with max depth 6 41</p> <p>6.6 Convergence of DO with the two-dimensional utility with one maximum and DT with max depth 7 41</p> <p>6.7 Convergence of DO with the utility with two maxima 42</p> <p>6.8 Convergence of DO with the three-dimensional linear utility and SVM with polynomial kernel with degree 6 44</p> <p>6.9 Convergence of DO with the three-dimensional utility with two maxima and SVM with polynomial kernel with degree 6 45</p> <p>6.10 Convergence of DO with the two-dimensional utility with one maximum and SVM with polynomial kernel with degree 4 45</p> <p>6.11 Time duration of training of SVM in seconds, depending on the utility function 46</p> <p>6.12 Convergence of DO with the two-dimensional linear utility and NN with 10 neurons in hidden layer, adding first better 47</p> <p>B.1 Solution times of discretization algorithm for scaled dimension count in seconds (player 1 dimensions \times player 2 dimensions) 61</p> |
|--|---|

Tables

| | |
|---|----|
| 5.1 Expressions for computation of numbers of points generated by each distribution in datasets | 26 |
| 5.2 The exact value of a Nash equilibrium computed by the specialized discretization | 27 |
| 5.3 The version of software used in the framework | 31 |
| 6.1 Results of experiments with the decision tree and the linear utility function. The columns are dimensions, the rows correspond to the maximal depth. | 39 |
| 6.2 Results of experiments with the decision tree and the utility function with one maximum. The columns are dimensions, the rows correspond to the maximal depth. | 39 |
| 6.3 Results of experiments with the decision tree and the utility function with two maxima. The columns are dimensions, the rows correspond to the maximal depth. | 39 |
| 6.4 Results of experiments with the SVM and the linear utility function. The columns are dimensions, the rows correspond to the degree of kernel. | 43 |
| 6.5 Results of experiments with the SVM and the utility function with one maximum. The columns are dimensions, the rows correspond to the degree of kernel. | 43 |
| 6.6 Results of experiments with the SVM and the utility function with two maxima. The columns are dimensions, the rows correspond to the degree of kernel. | 44 |
| 6.7 Results of experiments with the NN and the linear utility function. The columns are dimensions, the rows correspond to the number of neurons in hidden layer. | 47 |
| 6.8 Results of experiments with the NN and the utility function with one maximum. The columns are dimensions, the rows correspond to the number of neurons in hidden layer. | 48 |
| 6.9 Results of experiments with the NN and the utility function with two maxima. The columns are dimensions, the rows correspond to the number of neurons in hidden layer. | 48 |
| C.1 The version of software used in the framework | 63 |

Chapter 1

Introduction

1.1 Related Work

Machine learning algorithms, especially deep learning, are being applied in all branches of computer science these days. They can be easily used in various scenarios, and they can solve problems, which were unsolvable by traditional approaches in the past. Moreover, the convolutional neural networks often reach superhuman performance, which supports the expansion of machine learning methods even more. As typical examples of the usage, we can show applications in the computer vision [4, 5], in the voice recognition [6] or in the biological applications [7]. On top of it, machine learning is more and more used in the security [8, 9], where it is crucial to be robust against adversary attacks.

Unfortunately, machine learning methods are vulnerable to adversarial attacks. Especially, the deep neural networks are sensitive to it [10, 11]. But similar attacks were proposed also against other classifiers as SVMs [12] or linear statistical classifiers [13]. The adversaries typically distort the data to be misclassified. It is usually being demonstrated on images shown in Figure 1.1, but there also exist many examples of attacks in the security or the malware detection [14, 15].

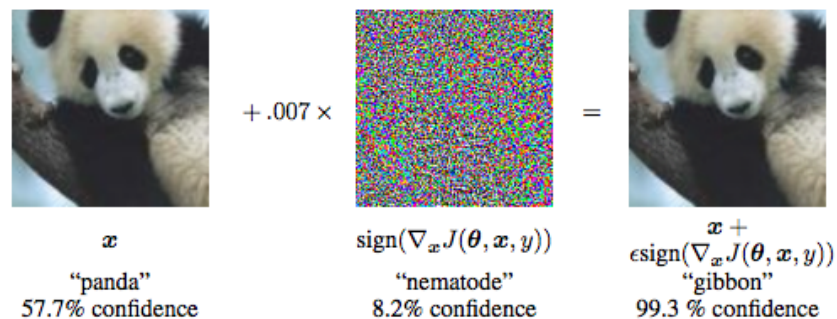


Figure 1.1: An example of the confusion of a classifier by addition of adversarial noise to the original image [1]

Adversarial attacks can lead to life-threatening situations. Thus, it is highly critical to find out a powerful classifier, which can stand against the attacks. In recent years, there have been many approaches to create more robust classifiers [16, 17, 18, 19, 20]. These defense strategies were proposed to be effective against some types of attacks, and none of them can be used as a universal defense [21]. It is because the classifiers give no guaranty on the quality of the solution [22]. Moreover, the majority of the algorithms for the training of robust deep neural networks is based on an obfuscating gradient, and there always exists an attack, which circumvents it [23].

It all points out the importance of the warranty of the robustness. We need to have a classifier with a guaranteed maximum loss after an attack. This leads us to game theoretic approaches. Field of game theory is specialized in finding optimal or approximately optimal strategies. For this purpose, it can model an adversarial learning problem to find the optimal classifier. However, most of the existing game theoretic models solve only simple problems with a static classifier. [24, 25, 26] Furthermore, the problems are limited by the properties of the utility function and the type of a classifier, so it is commonly not usable in real-world situations.

Finally, there are examples of the usage of game theory in large machine-learning problems. The first example is a reformulation of the Generative adversarial networks (GANs) to the Generative Adversarial Network Games (GANGs). GANs are well known adversarial settings, where two neural networks compete against each other. The adversarial one tries to generate data, which are not distinguishable from the original training set. The second network produces the likelihood that the attacker has generated the data [27]. In GANGs, this problem is reformulated as a two-player game, which is solved to find a Resource-Bounded Nash equilibrium as a robust solution. The second example, we mention, is the usage of Double-Oracle algorithm [28] in multiagent reinforcement learning, which performs with good results in large environments [29]. It gives us hope that game theory can employ algorithms for finding the guaranteed robust classifier for large problems.

1.2 Outline and Contributions

In this thesis, we have focused on the adversarial classification problem in the most general form. We have formalized the problem, and we have outlined some algorithmic approaches, how to find a robust classifier against the adversary.

In Chapter 2, we have defined some basic game concepts necessary for the problem formalization, and in Chapter 3, there are definitions of solution concepts as Nash equilibrium and basic algorithms to find them.

Chapter 4 focuses on the possibilities, how to formalize adversarial classification problem in terms of game theory and outlines some possibilities, how to solve it. There is also a discussion of described formalizations, which leads to a selection of the one used in the next parts.

After the formalism, Chapter 5 describes our framework for finding a Nash

equilibrium in the adversarial classification problems and all possibilities, how to use it.

Chapter 6 contains the main contribution of this work. There are described experiments, which map the space of all configurations and find out the difficulties of our chosen method.

The last Chapter 7 concludes all the results and proposes several ways, how to build on this work and a few potentially interesting connections to other works.

Chapter 2

Introduction to Game Theory

In this chapter, we will introduce the basic concepts of game theory. We will focus on discrete games in a normal form, which are the most fundamental part of game theory. Furthermore, we will generalize this concept to the games with a continuous strategy space called infinite games.

Game theory is designed for rational decision making under defined conditions. There are one or more players, which form a coalition (*cooperative games*) or compete (*non-cooperative games*). A *utility function* expresses the will of the players. It maps from all possible combinations of player's actions to the real numbers and represents the satisfaction of the player after the move.

The important factor in the game analysis is the correlation between utilities of players. When the sum of utilities of all players in all states is the same (*constant-sum games*), it is simpler to find of the solution. Especially, games, where all the utilities always sum to zero (*zero-sum games*), are analyzed much easier than games without any relation between player's utilities (*general-sum games*).

Generalization into infinite games causes several problems. There is an infinite space of possible actions, which makes the analysis more complicated. Moreover, optimal strategies may be infinitely large. Thus, we will introduce concrete subclasses of infinite games called continuous games and separable games. The restriction gives us a guarantee of the existence of optimal strategies and its finiteness.

2.1 Normal-Form Games

The normal-form game is the simplest concept of formalization of games. In these games, players move only once and simultaneously. A final reward or a penalty for a player comes from a combination of played actions.

Definition 2.1 (Normal-form game [30]). A (finite, n-person) *normal-form game* (NFG) is a tuple (N, A, u) , where:

- N is a finite set of n players, indexed by $i \in \{1, \dots, n\}$;
- $A = A_1 \times \dots \times A_n$, where A_i action is a finite set of actions available to player i . Each vector $a = (a_1, \dots, a_n) \in A$ is called an action profile;

$$u_i(\sigma) = \sum_{a \in A} u_i(a) \prod_{j=1}^n \sigma_j(a_j) \quad (2.1)$$

Example 2.6. As an example, we compute the expected utility in Rock, Paper, Scissors game for the first player and the strategy profile $\sigma = (\sigma_1, \sigma_2)$, where $\sigma_1 = (0, \frac{1}{2}, \frac{1}{2})$ and $\sigma_2 = (\frac{1}{2}, \frac{1}{2}, 0)$. It means, player 1 plays paper and scissors with the probability $\frac{1}{2}$ and player 2 plays only rock and paper with the same probability.

$$u_1(\sigma) = 1 \cdot \frac{1}{4} + 0 \cdot \frac{1}{4} - 1 \cdot \frac{1}{4} + 1 \cdot \frac{1}{4} = \frac{1}{4} \quad (2.2)$$

The expected utility for the first player is $\frac{1}{4}$. Moreover, Rock, Paper, Scissors is a two-player zero-sum game, which enables us to compute the expected utility for the second player: $u_2(\sigma) = -u_1(\sigma) = -\frac{1}{4}$.

The game analysis strongly depends on the properties of the game. In this work, we will focus on two-player games [30], because it corresponds to the adversarial classification problem, where there are one attacker and one defender.

The second important property is the relation between the player's utilities. The most general type is the *general-sum* game. There can be any relation between utilities. The analysis of this type of game is relatively complex. Thus, there exists a *constant-sum* game. Normal-form game is constant-sum, if $\sum_{i \in N} u_i(a) = c$ holds for all $a \in A$. A special case of constant-sum games is the *zero-sum* game, where $c = 0$. [30] The adversarial classification problem can be formalized as both, a general-sum game and a zero-sum game. In this work, we will focus mainly on the zero-sum representation. A deeper insight into this decision is in Chapter 4.

2.2 Infinite Games

The concept of finite normal-form games is not enough to describe all the problems. In our work, we need to formalize the problem, where the set of pure strategies is infinite and continuous. It leads to a generalized model of infinite games.

Definition 2.7 (Infinite game). A two-player *infinite game* is a tuple (N, C, u) , where:

- N is a set of two players, indexed by $i \in \{1, 2\}$
- $C = C_1 \times C_2$, where C_i is a compact metric space corresponding to the i^{th} player's set of pure strategies.
- $u = (u_1, u_2)$, where $u_i : C \rightarrow \mathbb{R}$ is a real-valued utility (or payoff) function for player i

Before the definition, we have talked about the generalization of normal-form games. Every finite space is a compact metric space under the discrete

metric. Thus, any finite normal-form game is also an infinite game under this definition. Another demonstrative example of an infinite game is a polynomial game. [31]

Definition 2.8 (Polynomial game [32]). A two-player infinite game is a *polynomial game*, if a set of actions C_i is a closed one-dimensional interval from \mathbb{R} , and utility u_i is a multivariate polynomial function $u_i : C \rightarrow \mathbb{R}$.

Example 2.9. For a demonstration, there is a simple two-player zero-sum polynomial game. The strategy space is $C = \{[0, 1], [0, 1]\}$. The utility for player 1 is a polynomial function $u_1(\sigma) = (\sigma_1 - \sigma_2)^2$. It is a zero-sum game, thus the utility for player 2 is $u_2 = -u_1$. Both utilities are shown in Figure 2.2.

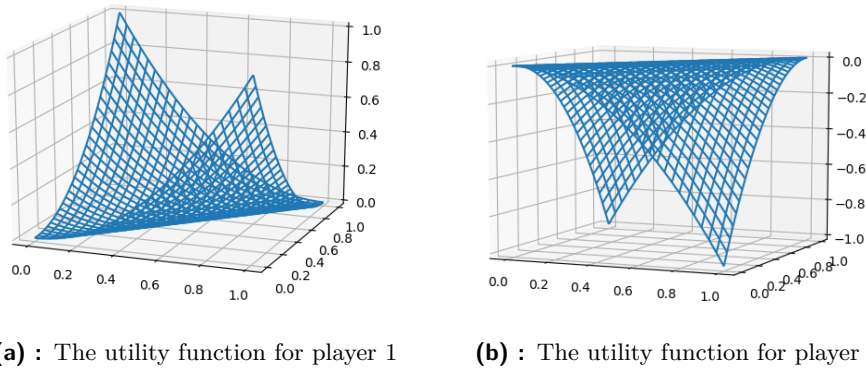


Figure 2.2: An example of a polynomial game

Similarly, as in the previous part, the next step is a definition of a mixed strategy and a strategy profile.

Definition 2.10 (Mixed strategy [33]). Let (N, C, u) be an infinite game, and for any set X let $\Delta(X)$ be the set of Borel probability measures over X . Then the set of *mixed strategies* for player i is $\Sigma_i = \Delta(C_i)$.

The set of mixed-strategy profiles is the Cartesian product of the individual mixed-strategy sets, $\Sigma_1 \times \Sigma_2$.

This definition of mixed strategies allows us to identify the pure strategy $s_i \in C_i$ with the atomic probability from $\Delta(C_i)$.

Example 2.11. Further to our example with the polynomial game, we choose a mixed strategy profile σ . In the σ , player 1 plays 0 with the probability of $\frac{1}{2}$ and 1 with the probability of $\frac{1}{2}$. Player 2 plays a pure strategy of $\frac{1}{2}$.

Eventually, we can define the expected utility for the player of the mixed strategy profile.

Definition 2.12 (Expected utility [33]). Given an infinite game (N, C, u) , the *expected utility* u_i for player i of the mixed-strategy profile $\sigma = (\sigma_1, \dots, \sigma_n)$ is defined as

$$u_i(\sigma) = \int_C u_i(c) d\sigma \quad (2.3)$$

Example 2.13. Following up on the example of the mixed strategy, we can compute the expected utility for the first player. The computation for strategy profiles with finite support is the same as computation for finite games.

$$u_1(\sigma) = \frac{1}{2} \cdot \frac{1}{4} + \frac{1}{2} \cdot \frac{1}{4} \tag{2.4}$$

Unfortunately, in the infinite games a solution in the form of Nash equilibrium does not need to exist. It leads us to the specification of the utility function and to the definition of continuous games.

Definition 2.14 (Continuous game [31]). A two-player *continuous game* is an infinite game (N, C, u) , where:

- $u = (u_1, \dots, u_n)$, where $u_i : C \rightarrow \mathbb{R}$ is a **continuous** real-valued utility (or payoff) function for player i

A typical example of a continuous game is a polynomial game because multivariate polynomial functions are continuous. Since every function defined on a discrete metric space is continuous, even a finite normal-form game is a continuous game. [31]

The concept of continuous games gives us a guarantee of the existence of a solution (Nash equilibrium). Unfortunately, the support of strategies in this solution can be infinitely large, which is practically unusable. Therefore, it is necessary to reduce the space of possible utility functions even more.

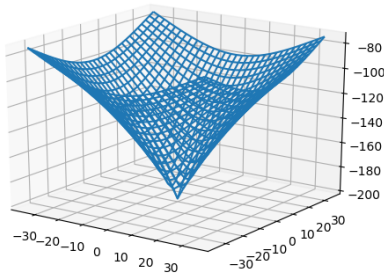
Definition 2.15 (Separable games [31]). A *separable game* is an infinite game with utility functions $u_i : C \rightarrow \mathbb{R}$ taking the form

$$u_i(\sigma) = \sum_{j_1=1}^{m_1} \dots \sum_{j_n=1}^{m_n} a_i^{j_1 \dots j_n} f_1^{j_1}(\sigma_1) \dots f_n^{j_n}(\sigma_n) \tag{2.5}$$

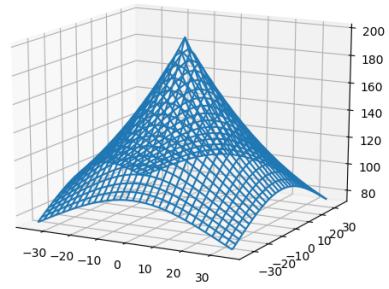
where $a_i^{j_1 \dots j_n} \in \mathbb{R}$ and the $f_i^j : C_i \rightarrow \mathbb{R}$ are continuous.

Separable game is a continuous game, and it can be seen as a generalization of a polynomial game, where different continuous function replaces the polynomials in one variable. Moreover, the finite normal-form games are also separable because f_i^j can be replaced by Kronecker delta function.[31]

Example 2.16. Let us demonstrate the difference between continuous games and separable games on an example of a two-player zero-sum non-separable game. The strategy space is $C = \{[-35, 35], [-35, 35]\}$. The utility for player 1 is a non-separable Ackley N.2 function $u_1(\sigma) = -200e^{-0.2\sqrt{\sigma_1^2 + \sigma_2^2}}$ [34]. The utility for player 2 is $u_2 = -u_1$. Both utilities are shown in Figure 2.3



(a) : The utility function for player 1



(b) : The utility function for player 2

Figure 2.3: An example of a non-separable game

Chapter 3

Solution Concepts

This chapter describes the stable and optimal strategies of rational players and the algorithms for exact or approximate computation of it.

The main focus of this work is on the Nash-equilibrium strategies and its finding in adversarial classification games. Thus, this chapter describes the standard algorithms for computation of a Nash equilibrium in normal-form games and in continuous games. The emphasis is given on solving zero-sum games, which are studied in this work. Further, there are mentioned algorithms for solving general-sum games. It helps us to understand different representations of adversarial classification problem as a game and supports our choice for experiments.

Eventually, we outline the concept of Stackelberg equilibrium, which is closely related to the adversarial classification problem and can give a meaningful alternative to Nash equilibrium. Moreover, some experiments confirm the Stackelberg character of this problem.

3.1 Nash Equilibrium

In games with more players, it is impossible to talk about the best strategy for one player, since the expected utility of the player depends on the strategies of the others. Thus, we need to talk about the best strategy against the strategies of all opponents. It is called the best response strategy.

Definition 3.1 (Best response [30]). Let $\sigma = (\sigma_1, \dots, \sigma_i, \dots, \sigma_n)$ be a strategy profile. Player i 's *best response* to the strategy profile $\sigma_{-i} = (\sigma_1, \dots, \sigma_{i-1}, \sigma_{i+1}, \dots, \sigma_n)$ is a mixed strategy $\sigma_i^* \in \Sigma_i$ such that $u_i(\sigma_i^*, \sigma_{-i}) \geq u_i(\sigma_i, \sigma_{-i})$ for all strategies $\sigma_i \in \Sigma_i$.

All the actions in support of the best response strategy have to have the same expected utility. Otherwise, the action with lower output would be dismissed to be played with zero probability. Therefore, the best response is a unique pure strategy, or there exist infinitely many of them. [30]

When the player plays the best response, he cannot gain more by changing his strategy. It leads us to the concept of Nash equilibrium, where all players play the best response. Thus, it is not beneficial to change the strategy for anyone.

Definition 3.2 (Nash equilibrium [31]). A mixed strategy profile σ is a *Nash equilibrium* if $u_i(\sigma'_i, \sigma_{-i}) \leq u_i(\sigma)$ for all i and all $\sigma'_i \in \Sigma_i$.

Example 3.3. The simplest example is a pure Nash equilibrium, where both players play a pure strategy. Let us have Prisoner’s dilemma game. There are two prisoners, which can either cooperate or defect. The utility is given by the matrix shown in Figure 3.1.

| | | Player 2 | |
|----------|-----------|-----------|--------|
| | | Cooperate | Defect |
| Player 1 | Cooperate | 3, 3 | 0, 5 |
| | Defect | 5, 0 | 1, 1 |

Figure 3.1: A payoff matrix for the Prisoner’s dilemma game [2]

The Nash equilibrium is the $\{defect, defect\}$ strategy profile. Both players gain utility 1 and by changing to cooperate they get only 0.

Example 3.4. In Rock, Paper, Scissors game, which is shown in the previous chapter in Table 2.1, there is one unique mixed-strategy Nash equilibrium $\sigma_1 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ and $\sigma_2 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. When one of the players changes his strategy, his expected utility stays the same.

This example shows that a pure strategy Nash equilibrium does not always exist, but in mixed strategies, a Nash equilibrium in a normal-form game exists all the time.[35] The proof is done by using Brouwer’s Fixed-point Theorem. [35, 36]

A Nash equilibrium, given by the definition, can also be found in infinite games.

Example 3.5. As an example, we can use the polynomial game from the previous section, shown in Figure 2.2. The Nash equilibrium in this game is a strategy profile, where player 1 plays 0 with the probability of $\frac{1}{2}$ and 1 with the probability of $\frac{1}{2}$. Player 2 plays a pure strategy of $\frac{1}{2}$. For neither one of the players, it is beneficial to switch to any other strategy.

Since Glicksberg generalizes the Fixed-point Theorem, the similar argument as for finite normal-form games proves the existence of a Nash equilibrium in continuous games.[37] Unfortunately, the absence of continuity of the utility allows us to create the game without a Nash equilibrium. Therefore, it does not need to exist in infinite games. The same importance has the condition of compactness of the strategy space.

Example 3.6. [31] This problem can be shown on a one-player game example. Consider a game with $C_1 = [0, 1]$ and a utility function:

$$u_1(x) = \begin{cases} x, & x < 1; \\ 0, & x = 1. \end{cases} \tag{3.1}$$

An inequality $u_1(\sigma_1) < 1$ for all strategies $\sigma_1 \in \Sigma_1$ holds, otherwise there

exists a point $S_1 \in C_1$, such that $u_1(s_1) \geq 1$. Thus, for every strategy $\sigma \in \Sigma_1$ we can find a point $s \in C_1$, such that $\sigma < s < 1$. Then $u_1(\sigma) < u_1(s)$, so there cannot be a Nash equilibrium.

Similar argument holds for a continuous utility defined in the same way on non-compact action space $C_1 = [0, 1)$.

Moreover, infinitely large strategy spaces can cause problems with an infinite size of the support of equilibrium strategies. It is problematic especially in the algorithmization and during the numeric computations. Thus, we have introduced separable games. The additional conditioning of utility function causes that all Nash equilibria in these games always have finite support. [38]

3.2 Finding of a NE in a Zero-Sum NFG

In the previous part, we have introduced the concept of Nash equilibrium and presented various guarantees for different types of games. In the next parts of this chapter are focus on algorithms for finding the exact Nash equilibria or its approximations in two-player games.

The first and most simple type is the zero-sum game. There holds that the loss of one player is a profit of the opponent. This intuition is confirmed by the von Neumann's Minmax Theorem [39, 30]. It shows that the utility U_1^* for player 1 is the same in all the Nash equilibria. This value is called the value of the game and can be solved by finding a minmax strategy of player 2. This result leads to the linear program. [30]

$$\text{minimize} \quad U_1^* \quad (3.2)$$

$$\text{subject to} \quad \sum_{k \in A_2} u_1(a_1^j, a_2^k) \cdot s_2^k \leq U_1^* \quad \forall j \in A_1 \quad (3.3)$$

$$\sum_{k \in A_2} s_2^k = 1 \quad (3.4)$$

$$s_2^k \geq 0 \quad \forall k \in A_2 \quad (3.5)$$

In a Nash equilibrium, all players play the best response, and in the best response all actions from the support have the same expected utility. Moreover, the actions, which are not in the support, have lower the expected utility. Therefore, the constraint (3.3) is tight in actions, which are in the support of the best response to the strategy profile s_2 . By minimization of U_1^* the program finds the optimal strategy of player 2 – the best response of player 1 has the lowest expected utility. The last two constraints only satisfy the conditions on the probability distribution.

Similarly, we can form an algorithm producing a strategy of the opponent. The players swap the role in the program, which has only two changes. Player 1 maximizes U_2^* , and the inequality sign in the constraint (3.3) has to be reversed.

$$\text{maximize} \quad U_2^* \quad (3.6)$$

$$\text{subject to} \quad \sum_{j \in A_1} u_2(a_1^j, a_2^k) \cdot s_1^j \geq U_2^* \quad \forall k \in A_2 \quad (3.7)$$

$$\sum_{j \in A_1} s_1^j = 1 \quad (3.8)$$

$$s_1^j \geq 0 \quad \forall j \in A_1 \quad (3.9)$$

This program corresponds to the dual of the first linear program [40]. It is great because we can get a Nash equilibrium strategy profile as a solution to a simple linear program and they are solvable in polynomial time [41].

3.3 Finding of a NE in a General-Sum NFG

General-sum games do not have directly competitive character. Thus, the solution cannot be expressed as a linear program. The most straightforward approach is to construct a linear complementarity problem [30] or it can be reformulated as a mixed integer linear program [42]. The most used approach to solve general-sum games is the Lemke-Howson algorithm [43].

Since the mixed integer linear programming is NP-hard, it can be expected that finding a Nash equilibrium in general-sum games is harder than finding it in zero-sum games. Christos Papadimitriou proves that the complexity of finding a Nash equilibrium in general-sum normal-form games is PPAD-complete [44]. Intuitively, it is faster than NP-complete because we know that the solution exists, but it is exponential in the number of actions. There is no proof whether the PPAD complexity class is not P or NP, but it is expected.

3.4 Finding of a NE in Infinite Games

There are not many works and computation results in this part of game theory. The first significant result is the generalization of the linear program for a two-player zero-sum game to zero-sum polynomial games [45]. This approach shows that it is possible to find a Nash equilibrium in infinite games efficiently. Unfortunately, the restriction of the game class does not allow the usage of the algorithm in real-world situations, where the polynomial cannot approximate the will of the players.

The second promising result is the generalization of a rank from NFG to separable games. The rank in normal-form games is identical with the rank of its utility matrix. With the notion of this term and results from [46], we can find a Nash equilibrium with support with the size of rank + 1 or smaller. It results in algorithms, which can approximate a Nash equilibrium in time polynomial in the rank. [38]

Unfortunately, there are no known algorithms for solving general infinite games.

3.5 Double-Oracle Algorithm

All the algorithms for solving NFG are usable only for small enough games. With the growing size of the game matrix, the duration of the computation grows fast. It creates a demand for an algorithm with better scalability. One of them is the iterative algorithm called Double Oracle. [28]

The basic idea of Double Oracle is simple. The algorithm solves multiple smaller games, which is faster than solving the LP or the MILP of the original game G . The computation starts with a restricted game G^r , where each player has only a subset of all possible actions. Usually, the subset is one action. This game is solved by one of the baseline methods introduced in the previous sections. It gives us strategies σ_1, σ_2 for both players. In the next step best responses are found to the strategies σ_1, σ_2 in the original game G . These best response actions expand the game G^r and all the procedure repeats. It is done until G^r does not contain the new best responses and stays the same after the expansion.

The main steps are visualized in Figure 3.2.

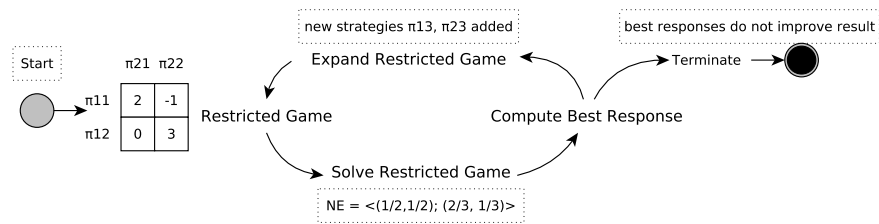


Figure 3.2: Schematic of the Double-Oracle algorithm. [3]

It is proven that this procedure always finds a Nash equilibrium. In the worst case, the algorithm appends all the actions into the restricted game, which leads to the significantly worse computation time than the LP or the Lemke-Howson algorithm. Fortunately, it is not usually the case. Commonly, the Double-Oracle algorithm terminates fast and expands a small subset of possible actions. [3, 28] Because of this conclusion, the convergence time is usually measured in the number of iterations of the algorithm.

This algorithm is usable in various scenarios and many types of practical problems. [29, 47, 48] It is useful even in the problems, where the exact Nash equilibrium is almost unreachable. We can use it for finding an approximation of the optimal solution. In the zero-sum games, the best responses calculated during the iteration give us an upper bound and a lower bound on the value of the game [28].

There exist some modifications of the main loop of the algorithm, which can lead to faster convergence. In the first modification, the players do not find the best response and expand the game in each iteration. The player, which will update the restricted game, is selected by some strategy. For example, they can alternate in the expansion [3]. The second change modifies the strategy to which the best response is found. There are usually being mixed the few last strategies according to predefined weights [29].

3.6 Stackelberg Equilibrium

The last topic outlined in this chapter is the Stackelberg equilibrium. It is a game-theoretical concept, where one player has a dominant position in the game. The dominant player (leader) commits his strategy, and the opponents react to it. The reaction is the best response to the strategy of the leader. [49]

Definition 3.7 (Stackelberg equilibrium [50]). Let (N, A, u) be a normal-form game, player 1 is the leader, and let Σ be a set of all mixed strategy profiles and $BR_i : \Sigma_1 \times \cdots \times \Sigma_{i-1} \times \Sigma_{i+1} \times \cdots \times \Sigma_n \rightarrow \Sigma_i$ returns the best response of player i . Then a mixed strategy profile σ is *Stackelberg equilibrium* if satisfies:

$$\operatorname{argmax}_{\sigma \in \Sigma; \forall i \in N \setminus \{1\}: \sigma_i \in BR_i(\sigma_{-i})} u_1(\sigma) \quad (3.10)$$

This solution concept can be useful in an adversarial classification problem because the defender commonly has the leader's position. He trains some classifiers, and the attacker tries to break them. In this work, we focus on the Nash equilibrium in an adversarial classification problem, but even there the Stackelberg character of this problem is significant.

Chapter 4

Adversarial Classification as a Game

In the previous chapters, we have introduced the basics of game theory. It explains the basic game models, solution concepts, and algorithms to find it. This chapter formalizes the adversarial classification problem in the game-theoretic framework and describes approaches used to find optimal defending strategies or its approximations.

Example 4.1. Let us begin with an example of an adversarial classification problem. We are managing a part of the computer network, and we want to create a security system, which detects a malicious traffic. We can measure only the size of the data transferred during one connection. Let us assume that the attacker tries to transfer as much data as possible. Thus, he has a utility, which linearly grows with the size of the data. When we detect the attack, we can stop it, and the attacker transfers nothing. Finally, we can limit the maximal payload on 10 GB for simplicity.

In this example, there are two players – the attacker and the defender. It holds for all adversarial classification problems. The adversary tries to play the strategy with the highest utility, and the defender tries to classify these points as attacker's.

Example 4.2. In our example, we can find a threshold, which divides the points into the benign on the left side and attacker's on the right side. Now we can write it as a game. The adversary selects the optimal size of transferred data, and the defender finds the optimal threshold. Both play a value from the interval $I = [0, 10]$. The attacker has utility $u_2(x) = x$. He gains it unless his point is not on the right side of the defender's threshold. We define this game as zero-sum. Both player's utility functions are shown in Figure 4.1.

Adversarial classification game is infinite since the space of the points and the space of the possible classifiers is infinite. Unfortunately, the game is not separable. The utility function of the attacker does not need to be from the class of separable functions. Thus, the support of a Nash equilibrium does not need to be finite. Moreover, our example demonstrates that the utility of the players is not continuous. Therefore, a Nash equilibrium does not need to exist in these games, because it can be in the one-side limit of the discontinuity as in Example 3.6.

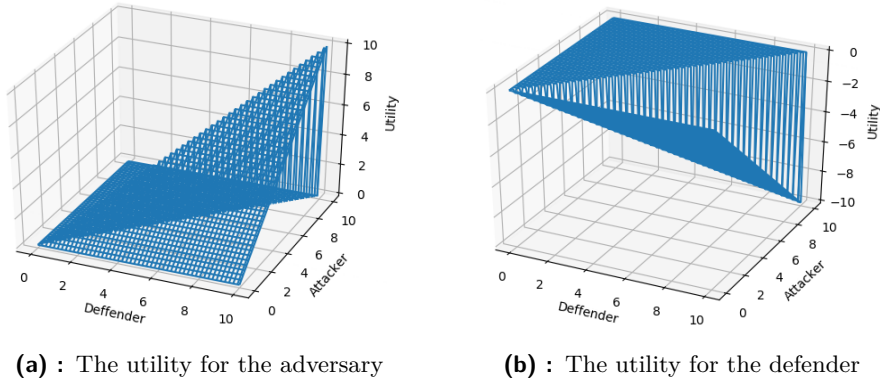


Figure 4.1: An example of an adversarial classification game. The utility function is linear until the diagonal, where the defender’s threshold is. Behind the diagonal there is utility 0. Vertical lines stand for the discontinuity.

4.1 General-Sum Game

The game defined in the previous section is trivial to solve. Since the defender can classify all the space as adversarial, the attacker cannot gain any profit. Therefore, his optimal strategy is to set the threshold on 0. The Nash equilibrium in this game is the strategy profile $(0,0)$. None of the players can get more than 0 by a change of the strategy.

Example 4.3. We can modify the game from Example 4.2 by adding data, which correspond to the regular traffic. We add two benign points at 3 and 5. When the defender misclassifies the point, he gets penalty equal to 1. The utility functions are shown in Figure 4.2, where the benign points are visualized as the black lines.

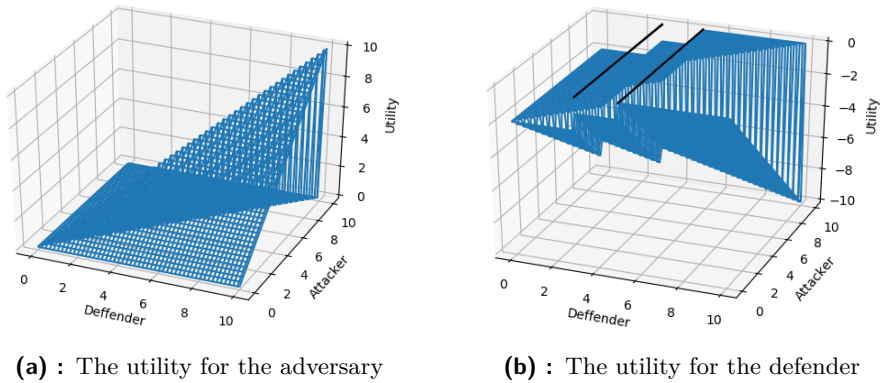


Figure 4.2: An example of an adversarial classification general-sum game. Black lines correspond to the benign points, and vertical lines stand for the discontinuity.

We can see that the game from Example 4.3 is general-sum. It is because the set of benign points reflects only the utility of the defender. The attacker’s utility is the same for any set of benign points.

Definition 4.4 (Adversarial classification game). *Adversarial classification game* is a two-player infinite game, where:

- $N = \{defender, attacker\}$, indexed by $i \in \{1, 2\}$
- $C = C_1 \times C_2$, where
 - C_1 is a compact metric space to the set of classifier's parameters
 - C_2 is a compact metric space to the set of attacker's pure strategies
- $P \subseteq C_2$ is a set of *benign points*
- $f : C_1 \times C_2 \rightarrow [0, 1]$ is a defender's *classification function*, where:
 - 0 corresponds to a benign point
 - 1 corresponds to an attacker's point
- $l : C_2 \rightarrow \mathbb{R}$ is a *loss function* for a benign point misclassification
- $u_2 : C_2 \rightarrow \mathbb{R}$ is an attacker's *default payoff function*
- $u = (u_1, u_2)$, where:
 - $u_2 : C_1 \times C_2 \rightarrow \mathbb{R}$ is defined as

$$u_2(c_1, c_2) = (1 - f(c_1, c_2)) \cdot u_2(c_2)$$

- $u_1 : C_1 \times C_2 \rightarrow \mathbb{R}$ is defined as

$$u_1(c_1, c_2) = -u_2(c_1, c_2) - \sum_{p \in P} f(c_1, c_2) \cdot l(p)$$

Example 4.5. Now, we can formally describe the previously developed game:

- $C_1 = C_2 = [0, 10]$
- $P = \{3, 5\}$
- $f : C_1 \times C_2 \rightarrow [0, 1]$ is:

$$f(c_1, c_2) = \begin{cases} 0, & c_2 < c_1; \\ 1, & c_2 \geq c_1 \end{cases}$$

- $l(p) = 1$
- $u_2(c_2) = c_2$

4.2 Finding a NE by Discretization

The first algorithm for the approximation of a Nash equilibrium is a simple discretization. We can sample the infinite strategy space to get a normal-form game. In this new game, it is not complicated to find a Nash equilibrium by algorithms from the previous chapter. The solution of the discretized game can be used as an approximation of a Nash equilibrium in the original infinite game.

Unfortunately, since the derivation of the utility function can be unbounded or even does not need to exist in some points, the density of the samples does not directly relate to the accuracy of the found solution.

The second and even more fundamental problem is the time complexity. The duration grows exponentially with the size of the game matrix. It

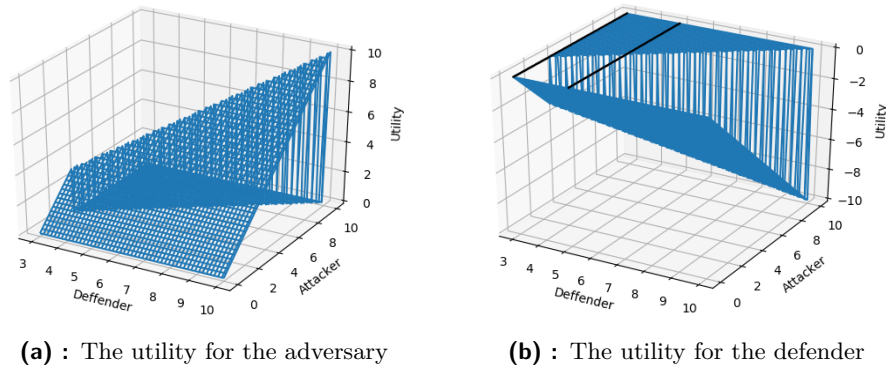


Figure 4.3: An example of a zero-sum adversarial classification game with a restricted false-positive rate of the classifiers

It forces us to reformulate the game as a zero-sum.

Only the loss for misclassification of benign points creates the game general-sum. To fix it, we need to consider the negative points in another way. There are three approaches: 1) hard constraint in classification, 2) hard constraint in expectation, and 3) soft constraint in expectation. They limit or minimize a false-positive rate of the defender’s strategies and make the game model zero-sum or almost zero-sum. A detailed explanation is in the following sections. Finally, the false-positive rate restriction creates the model useful in security applications, where the benign data almost should not be misclassified.

4.4.1 Hard Constraint in Classification

The first alternative is to limit the pure-strategy space of the defender strictly. We can use only classifiers, which have limited the false-positive rate. It restricts the strategies, which are strongly influenced by the penalty for the misclassification of the benign points. Moreover, it creates a meaningful zero-sum variant of the problem formalization.

Example 4.6. Let us continue with the game from Example 4.3. We formulate it as a zero-sum game ($l(p) = 0$), and we restrict classifiers by the false-positive rate equal to 0.5. It allows classifying one benign point wrongly. Utilities are plotted in Figure 4.3.

Since the restriction of the false-positive rate in the classifier training is not always crucial, there only exists a small number of classifiers, which support this condition, and they are not usually the most used state of the art classifiers. Therefore, we do not focus on this game representation.

4.4.2 Hard Constraint in Expectation

The second possibility, how to bring the negative data into the zero-sum adversarial classification game, is to constrain the mixed strategies by the false-positive rate. It does not affect the space of pure strategies, but it allows

players to play strategies with a high false-positive rate only with a small probability. It means, the worst expected false-positive rate is the same as in the previous solution, but sometimes, we can play a strategy, which in general misclassifies a lot of benign points.

A game from our definition is not purely zero-sum or even in the normal form, but we can find a Nash equilibrium by a small modification of the linear program.

$$\text{minimize } U_1^* \quad (4.1)$$

$$\text{subject to } \sum_{k \in A_2} u_1(a_1^j, a_2^k) \cdot s_2^k \leq U_1^* \quad \forall j \in A_1 \quad (4.2)$$

$$\sum_{k \in A_2} s_2^k \cdot fp(a_2^k) \leq FP \quad (4.3)$$

$$\sum_{k \in A_2} s_2^k = 1 \quad (4.4)$$

$$s_2^k \geq 0 \quad \forall k \in A_2 \quad (4.5)$$

The linear program is expanded by the constraint (4.3). FP is constant limiting the overall false-positive rate and $fp(a_2^j)$ is the false-positive rate of the action a_2^j . A Nash equilibrium strategy for the opponent is gained as a solution to the dual problem to this LP.

In this work, we evaluate this variant, because it can be used with the most popular classifiers these days. Unfortunately, there can be some complications with using this in Double Oracle. First of all, it is necessary to start the algorithm with a restricted game, which contains a classifier classifying everything as benign. Otherwise, the linear program can be infeasible.

Example 4.7. For example, we have the following game:

- $C_1 = C_2 = [-5, 5]$
- $P = \{-2, 2\}$
- $f : C_1 \times C_2 \rightarrow [0, 1]$ is:

$$f(c_1, c_2) = \begin{cases} 0, & c_2 < c_1; \\ 1, & c_2 \geq c_1 \end{cases}$$

- $u_2(c_2) = c_2^2$
- $FP = 0.1$

Attacker's initial point is 0. The points $\{-2, 2\}$ and $\{0\}$ are not separable. Thus, without loss of generality, the best response classifier classifies the point $\{-2\}$ as benign and points $\{0, 2\}$ as adversarial. The restricted game contains adversarial point 0 and the classifier, which has a false positive rate higher than FP . Linear program for this game is infeasible.

The second and more severe problem is that the classifier found by some training algorithm does not need to correspond to the best response strategy

in the game with false-positive constraint in expectation. The classifier can find a strategy with a high false-positive rate and with a high utility, but there can exist a slightly worse classifier with a much lower false-positive rate. Thus, in the final game, the second classifier brings much more to the expected utility than the first one. Moreover, the best response strategy does not need to exist there, in pure strategies, and in some cases, it would be necessary to find a mixed one.

4.4.3 Soft Constraint in Expectation

The last option of reduction of the false-positive rate is the soft constraint in expectation. It is done by moving the constraint from the previous subsection to the objective function of the linear program.

$$\text{minimize} \quad U_1^* - C \cdot \sum_{j \in A_2} s_2^j \cdot fp(a_2^j) \quad (4.6)$$

$$\text{subject to} \quad \sum_{k \in A_2} u_1(a_1^j, a_2^k) \cdot s_2^k \leq U_1^* \quad \forall j \in A_1 \quad (4.7)$$

$$\sum_{k \in A_2} s_2^k = 1 \quad (4.8)$$

$$s_2^k \geq 0 \quad \forall k \in A_2 \quad (4.9)$$

This modification does not strictly limit the false-positive rate of the final solution, but it only prefers strategies with the lower number of false positives. It is great because we do not need to set the threshold on the false-positive rate. However, there is another constant C , which balances the utility and the false-positive regret. Unluckily, this constant can be hardly chosen without tests or good insight into the concrete problem. Therefore, we have left this option out, and we have evaluated only the previous one.

4.5 Specialized Discretization for Game with Hard Constraint in Expectation

The zero-sum adversarial classification game representations with false-positive constraint in expectation allow us to introduce a simplified discretization algorithm. The attacker divides his strategy space into intervals. These intervals correspond to the strategies in the discretized game. The defender uses the same intervals for the classification. To each interval, he sets a probability, that he classifies it as adversarial. The probabilities are constrained only by the constraint on false-positives. The following LP can solve this task:

$$\text{minimize} \quad U_1^* \quad (4.10)$$

$$\text{subject to} \quad u_1(s_2^j) \cdot (1 - s_2^j) \leq U_1^* \quad \forall j \in I \quad (4.11)$$

$$\sum_{j \in I} s_2^j \cdot fp(j) \leq FP \quad (4.12)$$

$$1 \geq s_2^j \geq 0 \quad \forall j \in I \quad (4.13)$$

The set I is a set of intervals, s_2^j is the probability of the classification of interval j as adversarial, $u_1(s)$ is the attacker's utility gained from the interval s , and $fp(j)$ is a false-positive rate of interval j .

When we look at the linear program deeper, we can see that s_2^j is always 1 for intervals with $fp(j) = 0$. Thus, we can leave them out and rapidly reduce the size of the linear program. Owing to this trick, we can increase the density, and the linear program grows slowly. When we set the density to infinity, the intervals exactly correspond to the points in the space. In this case, only the benign points have nonzero fp rate. Therefore, we can identify the set I in the linear program with the set of benign points and $fp(j) = \frac{1}{|S|}$. In these settings, the linear program computes the exact Nash equilibrium because the density cannot be higher.

We have an algorithm for finding the exact Nash equilibrium in adversarial classification problem with hard constraint in expectation, which is polynomial in the size of the set of benign points. Unfortunately, this method overfits the data. Therefore, it is not usable for real-world problems.

Chapter 5

Framework Details and Setting of Experiments

The previous sections describes the basics of game theory and its application to solve an adversarial classification problem. We have decided to use the zero-sum game representation with hard constraint in expectation. It allows us to get the exact value by specialized discretization and use the standard classification methods in Double Oracle.

5.1 Datasets

The datasets, which we have produced for the experiments, have two parameters: 1) the number of dimensions of the space of the attacker, and 2) the number of local maxima of the attacker's utility function. Thus, we have introduced three functions parametric in the number of dimensions, where the first one is linear in all dimensions, the second one has one local maximum in the middle of the interesting interval, and the third one has two local maxima on the diagonal in all dimensions. All these functions are defined on intervals $[0, 10]$ in each dimension, and the range is normalized on the interval $[0, 10]$.

$$f(x) = \frac{1}{dim} \sum_{n=i}^{dim} x_i \quad (\text{Linear function})$$

$$f(x) = 10 - \frac{\sum_{n=i}^{dim} (x_i - 5)^2}{dim \cdot \frac{5}{2}} \quad (\text{Function with one maximum})$$

$$f(x) = \max \left\{ -\frac{\sum_{n=i}^{dim} ((x_i - 5) \cdot dim)^4 - 25 \cdot dim \cdot \left(\sum_{n=i}^{dim} (x_i - 5) \cdot dim \right)^2}{\frac{625}{40} \cdot dim^5}, 0 \right\} \quad (\text{Function with two maxima})$$

These functions are displayed for dimensions 1 and 2 in Figure 5.1.

The benign data are generated randomly form normal distributions corresponding to the lower values of the utility function. The first and the third one

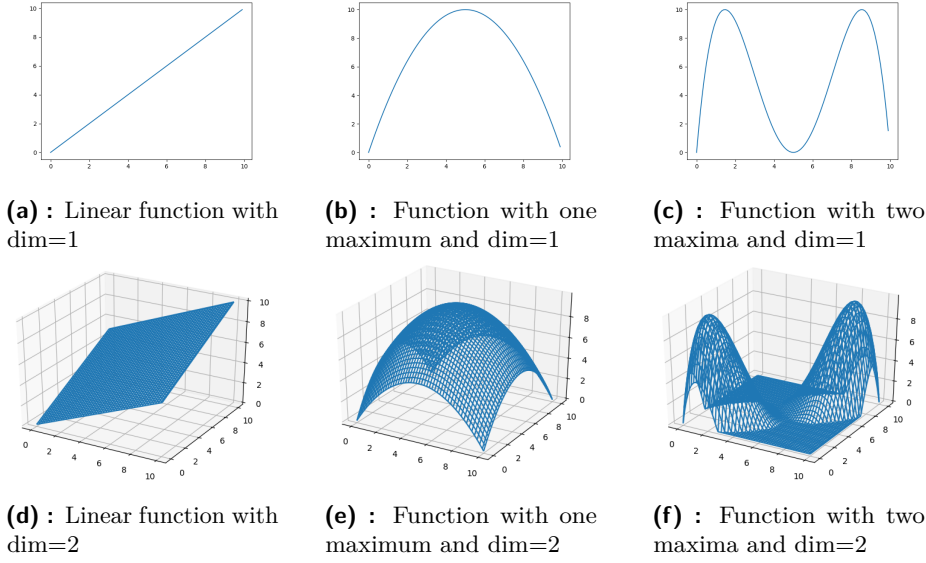


Figure 5.1: The functions displayed for dimensions 1 and 2

have points generated only from one distribution, and points corresponding to the second function are generated from $1 + dim$ normal distributions around the local maximum. All the points are forced to be on the intervals $[0, 10]$.

The first dataset is generated from normal distribution with mean $\mu = \vec{1}$ and covariance matrix $\Sigma = \mathcal{I}$. The third dataset is generated from normal distribution with mean $\mu = \vec{5}$ and covariance matrix $\Sigma = 9 \cdot \mathcal{I} + (\frac{-9}{dim-1} + 0.5) \cdot (1 - \mathcal{I})$. For the one-dimensional case the covariance matrix is $\Sigma = \mathcal{I} \cdot 0.5$. The second dataset is generated from normal distributions with equal covariance matrices $\Sigma_i = \mathcal{I}$ and means $\mu_1 = [1, 1, \dots]$, $\mu_2 = [9, 1, \dots]$, $\mu_3 = [1, 9, \dots]$, etc. The number of points generated from each distribution is calculated using equations written in Table 5.1.

| Dataset 1 | Dataset 2 | Dataset 3 |
|---------------------------|--------------------------|----------------------------|
| $10 + 40 \cdot (dim - 1)$ | $5 + 30 \cdot (dim - 1)$ | $5 + 50 \cdot (dim - 1)^2$ |

Table 5.1: Expressions for computation of numbers of points generated by each distribution in datasets

The datasets generated for dimension 1 and 2 are displayed in Figure 5.2.

These functions and datasets cover all interesting situations, which can be solved. The linear utility is the most straightforward situation. The defender needs to separate two of the most distant points. The function with one local maximum complicates the situation because the benign points almost surround the attacker's optimum. Thus, it has higher demands on the classifier. Finally, the third utility function switches the situation, and the attacker's maxima surround the benign points.

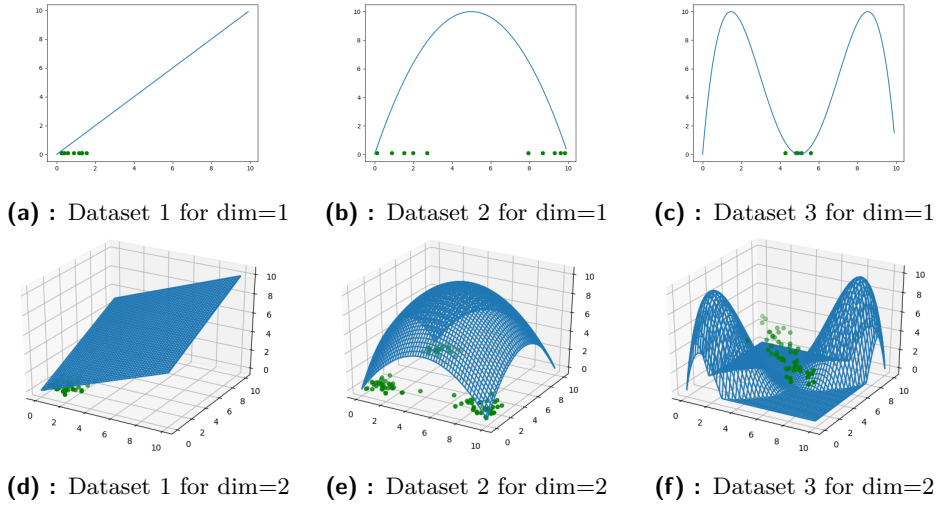


Figure 5.2: The datasets displayed for dimensions 1 and 2

5.2 Specialized Discretization

First of all, we run the exact specialized discretization on introduced functions and datasets. It gives us the exact value of a Nash equilibrium with optimal classifier. We have used the FP threshold equal to 0.01. The results are shown in Table 5.2.

| dim 1 | Linear | 1 maximum | 2 maxima |
|-------|---------|-----------|----------|
| 1 | 1,39609 | 7,11946 | 0,79075 |
| 2 | 2,00348 | 6,22406 | 0,92138 |
| 3 | 1,84211 | 5,95752 | 0,87073 |
| 4 | 1,84884 | 5,73538 | 0,43125 |
| 5 | 1,71698 | 5,27989 | 0,54156 |

Table 5.2: The exact value of a Nash equilibrium computed by the specialized discretization

All the runs have ended almost immediately. The longest time was necessary for the second function in five dimensions, where the average from 20 runs has taken 1,051 s, and the standard deviation has been 0.029. Unfortunately, the final strategy is unusable in real situations, because this algorithm overfits the set of benign points.

5.3 Double Oracle Setup

The previous chapters describe the main loop of the Double-Oracle algorithm and its application on the adversarial classification problem. Now, we will focus on possibilities, how to implement the best response algorithms.

In the adversarial classification problem, we can interpret Double Oracle

as follows. In each iteration, the attacker generates new data point, for which the actual utility is maximum, and the defender finds the best classifier to separate the currently played attacker’s data from the benign ones. When the algorithm solves the linear program, it changes the currently played attacker’s points and currently played classifiers, which are reflected on the actual utility. When no new point and no new classifier is added, we have the optimum in the predefined settings.

5.3.1 Attacker’s Best Response

The attacker optimizes the actual utility, which corresponds to the initial utility multiplied by the classification, weighted by probabilities of individual classifiers. The benign points have to be classified as 1, and the adversarial points have to be classified as 0. Thus, we need to find the global optimum of a discontinuous function. There exist standard methods to solve it.

Basin-Hopping

In our implementations, we select Basin-Hopping global optimization. This stochastic method uses surface transformation into the energy function. Then the algorithm randomly changes the coordinates and runs the local gradient optimization [51]. We have used this method because it performs with good results, and there exists a package with the standard implementation. For the local gradient optimization, we have used the L-BFGS-B algorithm [52].

The usage of this algorithm has a disadvantage. During the computation, it calls the optimized function many times. In our problem, each call needs to classify the point by all the classifiers. It is proven to be a distinctive bottleneck of the algorithm.

Basin-Hopping with Discretization

The first improvement is to start with a reasonable estimation of the maximum. This step can cut off some evaluations of the utility function at the beginning of the computation.

We can compute a value of the utility function in points from a grid. In each iteration of Double Oracle, we need to save the classification of the points by the newly added classifier. The optimization algorithm starts with a multiplication of the utilities, with the classification and the actual support of the defender. It gives us the actual utility in all points from the raster, and by a simple selection, we can choose a good starting point for the optimization algorithm.

We select multiple points and use them as a starting point for a parallel run of the Basin-Hopping algorithm. Unfortunately, in some runs, the stochastic optimization evaluates the utility function between the precomputed points many times.

■ Discretization with a Gradient Optimization

The speedup of the algorithm can be done by replacement of the stochastic optimization by a gradient method. This exchange removes the correctness of the algorithm - the found value is only an approximation of the maximal value. On the other side, this modification speeds up the computation, because the gradient optimization needs fewer evaluations of the function.

The discretization causes complications. To save the points, the demands on the memory grow exponentially with the dimension of the discretized space. For example, when we use 100 samples in each dimension, the five-dimensional table contains 100^5 points. When we use 4 bytes to store the coordinates of the point, we need $4 \cdot 5 \cdot 10^{10} = 2 \cdot 10^{11}$ bytes ~ 200 Gigabytes.

The exponential memory complexity forces us to the trade-off between the memory needed for caching points and a longer run of the optimization function with a higher risk of error.

The gradient optimization algorithm used in this work is the L-BFGS-B algorithm [52].

■ Additional improvements

In the adversarial classification problem, benign points play an essential role in the solution. Thus, the convergence of the Double-Oracle algorithm could be sensitive to the exact results of the optimization in the neighborhood of these points. Unfortunately, all the algorithms have numerical inaccuracies, which can lead to the incorrect convergence of Double Oracle. Therefore, we add the benign points to the set of discretized points.

■ 5.3.2 Defender's Best Response

The attacker generates new points as the best response. The defender reacts by finding a classifier. The expected value of classifier depends on the utility and the probability of playing of misclassified attacker's points. Thus, we need to find classifiers, which minimizes weights of misclassified attacker's points with respect to the number of misclassified number of benign points. It leads us to the usage of the standard classification methods, which supports weighting of training points.

There is a question, how to weight the benign points during the classification to get the best approximation of the best response classifier. We tried two variants: 1) all benign points have weight 1, 2) benign points has weight $\frac{1}{n}$, where n is a count of the benign points.

In this work, we start with the decision-tree algorithm, which is the simplest concept of the classifiers. Then, we move to the Support Vector Machines, which represent classifiers with a simple structure - linear or polynomial classifiers, and it can give reasonable results in real-world problems. The last examined classifier is the neural network, which is the most used classifier these days.

■ Decision Tree

The depth of decision tree is a parameter, which determines the strength of the algorithm. We have tested three possible settings. The first one limits the maximal depth of the tree, and the training is based on the impurity. In the second approach, we check the expected utility of the tree in each depth change. When the classifier is slightly better than all classifiers that already are in the restricted game, we use it as the best response. The third one also checks the expected utility in each depth, but the tree is being extended until the weighted change of misclassified points is higher than the threshold. The second and the third approach has not limited the maximal depth.

■ Support Vector Machine

We have used the support vector machine with a polynomial kernel. There is the degree of the kernel, the parameter, which determines the complexity of the classifier. In the SVM, it is necessary to set there the C parameter, which makes a trade-off between the misclassification penalty and complexity of the classifier. In Double Oracle, we need to separate the attacker's points exactly – they cannot be overfitted. Thus we set the $C = 10000$, which corresponds to the huge penalty for misclassification, compared to the penalty for the classifier structure gap.

■ Neural Network

The tested neural network is a multilayer perceptron with one hidden layer. The input layer has the number of neurons corresponding to the number of dimensions of the attacker's space. The number of neurons in the hidden layers is the parameter determining the strength of the classifier. The structure is shown in Figure 5.3 The loss function used in training is the weighted cross entropy. There are two options when the learning is ended. The first option is that the classifier is used when its expected utility is higher by the threshold than the expected utility of all previous classifiers. The second option continues learning until the change of the loss is smaller than the defined constant.

In each iteration of Double Oracle, we use the neural network from previous learning as the initialization of weights.

■ 5.4 Implementation Details

We have implemented a framework for the usage of Double Oracle in adversarial classification problem. It is based on a simple modification of classifiers, utility functions, and attacker's best-response optimizers. The main loop is written in Python 3, using NumPy. The classifier, optimizer, utility function, and benign points enter the main loop as parameters.

Based on the related work [3, 29], the main loop has three alternatives: 1) the best responses are calculated simultaneously, 2) players alternate in

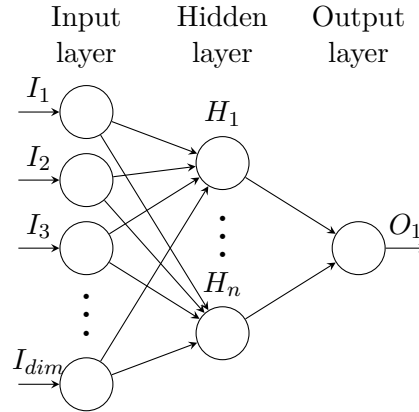


Figure 5.3: Schema of the used neural network

the calculation of the best responses, 3) the best responses to the mixture of last few strategies are calculated simultaneously.

Before the calculation of the best responses, the framework solves the LP. It can be done by implemented Gurobi solver or by using CVXOPT solver. The linear program is prepared for solving games with hard constraint in expectation and with hard constraint in classification. The chosen option depends on the user's preferences.

In the framework, there are prepared two options of attacker's best-response optimizers: 1) Basin-Hoping with discretization, and 2) Discretization with local L-BFGS-B optimization. They are implemented according to the description in Section 5.3.1. However, the user can simply replace the optimizer by others, for example, the default Basin-Hoping stochastic optimization. All the algorithms are using SciPy.

There are implemented three classifier options: 1) SVM classifier using SVC classifier from scikit-learn, 2) decision tree using the implementation from scikit-learn, and 3) neural networks using the implementation from PyTorch. All the classifiers are implemented according to the description in Section 5.3.2

The versions of used libraries are listed in Table 5.3. Gurobi is used via its Python interface called GurobiPy.

| Library | version |
|---------------|---------|
| Python: | 3.6.2 |
| NumPy: | 1.16.1 |
| SciPy: | 1.1.0 |
| Gurobi: | 7.5.2 |
| CVXOPT: | 1.2.3 |
| scikit-learn: | 0.19.1 |
| PyTorch: | 0.3.0 |
| matplotlib: | 3.0.3 |

Table 5.3: The version of software used in the framework

More implementation details are in Appendix C.

Chapter 6

Experiments

In Chapter 5, we have described the Double-Oracle framework for an adversarial classification problem. Chapter 6 describes the experiments performed on this framework. The experiments have been performed to evaluate the usage of Double Oracle in adversarial classification problem with various settings. We have evaluated the simultaneous and alternating computation of player's best response in the main loop of the algorithm, the multiple optimization functions to get the attacker's best response, and classifiers with different settings of parameters. We have performed experiments with the decision tree classifier, with SVM, and with a neural network. All the experiments have used the game representation with a hard false-positive constraint in expectation with FP threshold equal to 0.01. The algorithm has ended, when the difference between the actual utility of the best response and actual utilities of all actions in the restricted game was smaller than 0.5 % of the maximum of the utility function.

6.1 Alternating or Simultaneous BR Computation

The Double-Oracle algorithm has multiple options, how to form the main loop of the algorithm. In the first and the original setting, players calculate the best response in the same time, and each iteration is the game expanded in both dimensions – in the dimension of the attacker and the dimension of the defender. In the second option, the players alternate in the calculation of the best response. The first experiments expose the difference between these options in an adversarial classification problem.

We have performed an experiment with two-dimensional utility function with two local maxima and the third dataset of benign points. We have used the discretization containing 100 samples in each dimension and with a local gradient optimization as the attacker's best-response function. Then we have used the SVM classifier with a polynomial kernel with degree 2, and with the weights of benign points equal to 1.

The run with the simultaneous BR computations has ended after 104 iterations with value 0.92138, which is almost equal to the optimal value. In contrast, the run with alternating computations of BR has ended after 8 iterations with value 4.99999.

The difference is demonstrated in the first few iterations which are plotted in Figures 6.1 and 6.2. The benign points have a green color, newly added point and classifier have a yellow color, classifiers in the support are green, points in the support are magenta, the other points are red, and other classifiers are blue. The height of the depiction corresponds to the probability of playing. The initial classifier classifies all the space as benign, and it is not displayed.

When the computation of BR alternates, the classifier always reacts only on one adversarial point. In the fourth iteration, the attacker and the defender mix up between two points and classifiers, but on this strategy only the attacker interacts. The defender computes the best response only against the newly added point. In the simultaneous version, both players interact with

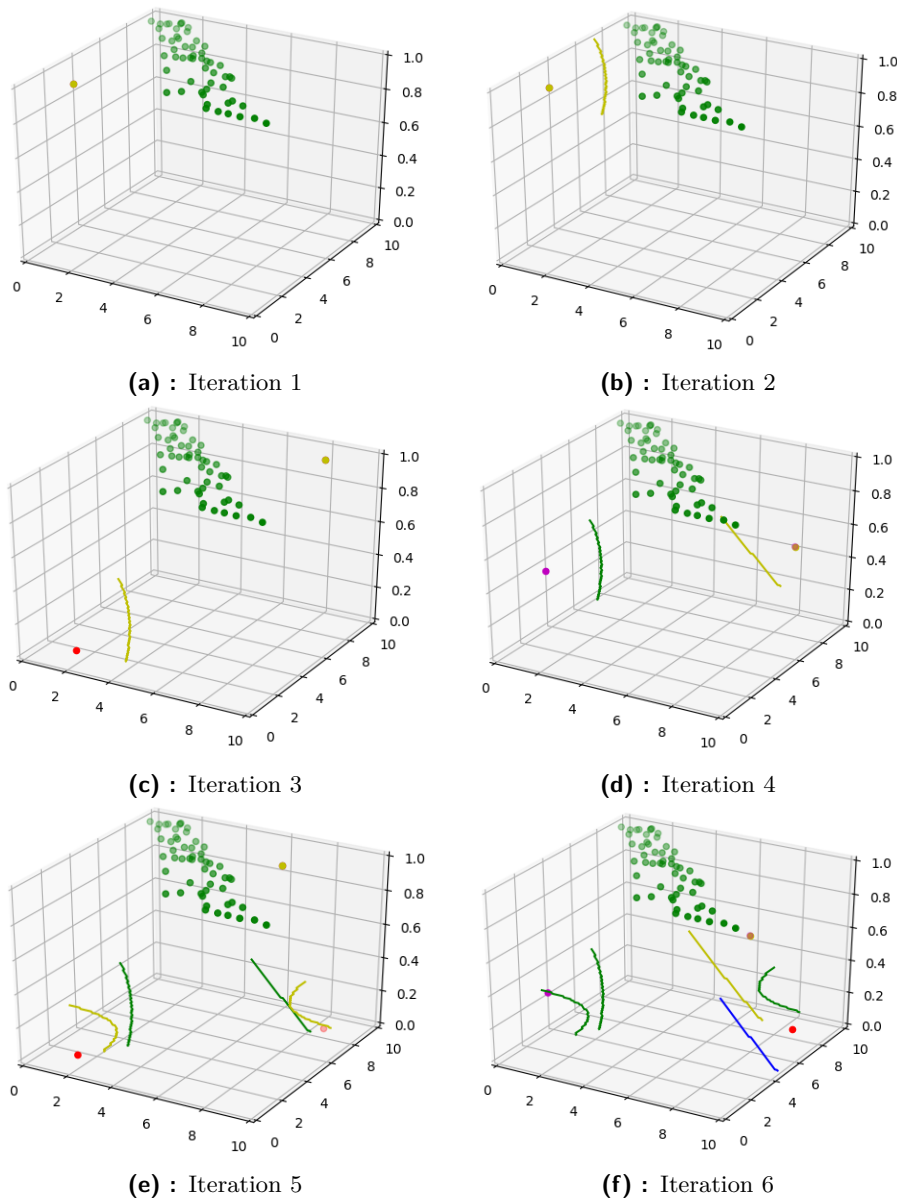


Figure 6.1: A few iterations of the run with simultaneous BR computation

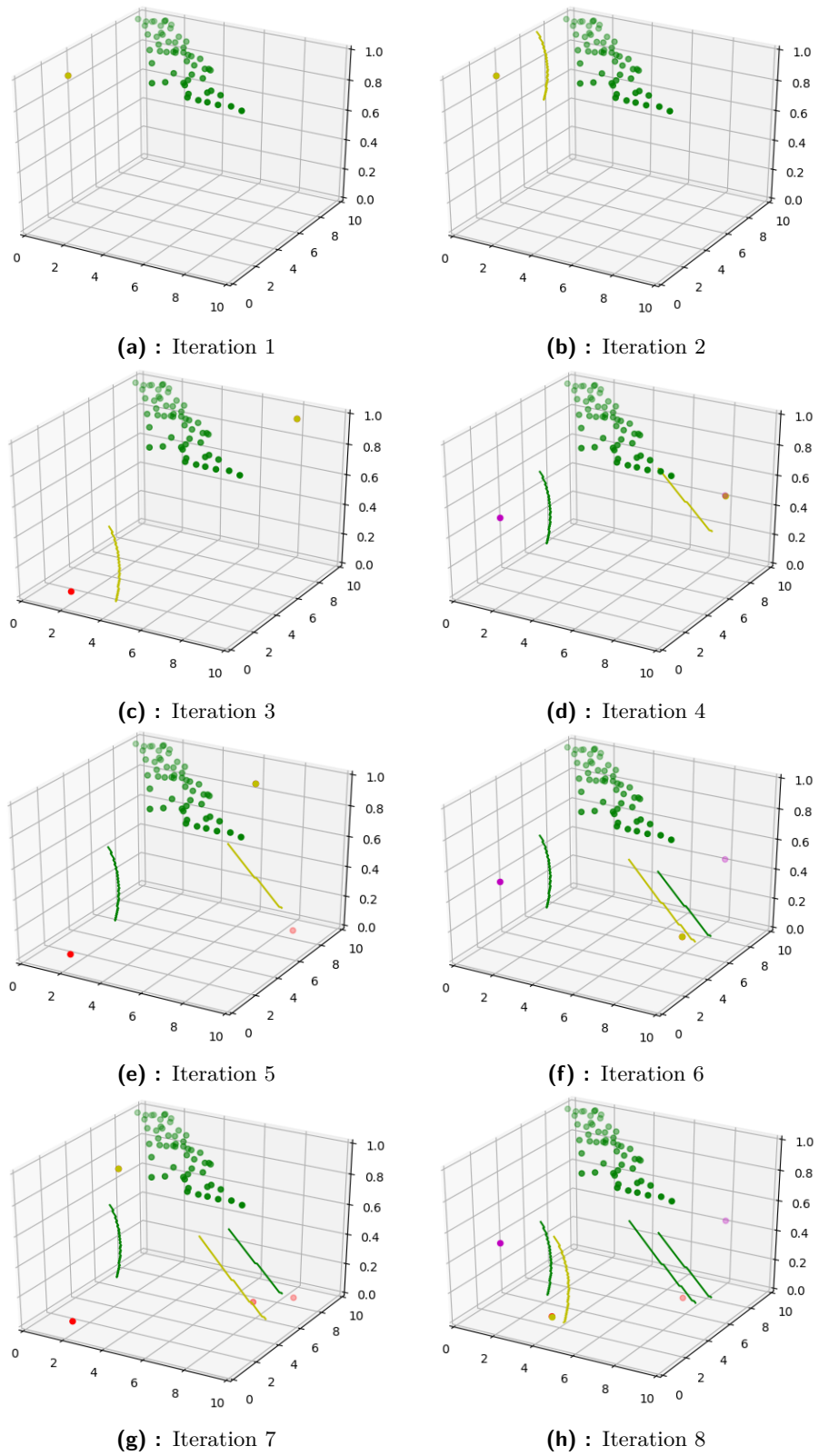


Figure 6.2: A few iterations of the run with alternating BR computation

the strategy, which mixes between points or classifiers on both sides of benign points. Therefore, in the restricted game, there are classifiers, which separate both sides at once.

Because of the robustness, we have decided to use the simultaneous computation of the best response in all the experiments. However, in some cases, the alternating computation can speed up the algorithm. For example, in the experiment with the two-dimensional linear utility and with the first dataset, the alternating computation of best-responses accelerates the run of the algorithm by almost a third. It has about a quarter more iterations (122 vs. 90), but the alternating ones need only a half of BR computations.

6.2 Comparison of Optimization Algorithms

In our framework, there are three possible optimization methods: 1) Basin-Hopping, 2) Basin-Hopping with initial discretization, and 3) discretization with gradient optimization. In this section, we compare the performance of these three optimization methods.

The tests have run with the two-dimensional linear utility function and the first dataset of benign points. The defender's best response has been calculated by SVM classifier with a polynomial kernel with degree 2. The benign points are weighted by $\frac{1}{n}$, and the discretization algorithms use a grid with 100 points in each dimension. The results plotted in Figure 6.3 display the average time of all the runs. The time is summed up over all iterations. The standard deviation of all experiments is smaller than 5 %.

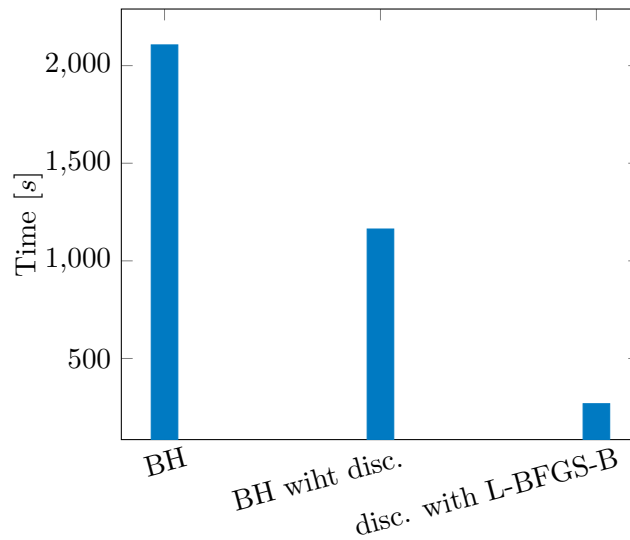


Figure 6.3: Time duration of the optimization algorithms

We have repeated the experiments with the same settings to evaluate the error of discretization with L-BFGS-B compared to the discretization algorithm with Basin-Hopping. The maximal error of all runs was 0.0635. The average maximal error of all runs was 0.0482 with the standard deviation

equal to 0.0095. Since the errors are smaller than 1 % of the maximum utility, we can consider them negligible.

Based on these experiments, we have used the discretization algorithm with L-BFGS-B in all later experiments.

6.2.1 Insertion of Benign Points to the Optimization

As an improvement of the optimization algorithm, we have inserted the benign points to it as a baseline value of the result. We have performed two equal experiments to demonstrate the impact on the convergence of Double Oracle. We have used the one-dimensional linear utility function and the first dataset of benign points, and the decision tree classifier with maximal depth 3. The optimization function has been the discretization with L-BFGS-B, with 100 samples.

Double Oracle with benign points added to the optimization converges after 41 iterations to the value 1.39609, which exactly corresponds to the optimal value calculated by the specialized discretization.

Double Oracle without benign points added to the optimization converges after 169 iterations to the value 1.05222, which is lower than the optimal value.

By examining of the computation course, we can find out a problem in the second run, when the best responses approach to the benign point with the highest utility, equal to 1.5512166. The attacker's optimizer finds the point 1.55124279, which is slightly higher than the benign one. The decision tree precisely crops it at 1.55122995. In the next iteration, the optimizer does not find the optimal value, but only the point 1.55121551. In the next step, the decision tree finds a classifier with cuts in 1.43356001, 1.55121648, and 1.55122995. The interval between the first two cutting points is classified as adversarial and every point higher than the third point is also adversarial. This classifier has false-positive rate 0, and it correctly classifies all added attacker's points. Moreover, the gap between the second and the third cutting point cannot be found by the optimization, although the maximum of the utility function is in it. Since the benign point with the highest utility is in this interval, it is a better approximation of the attacker's best response than the value given by the discretization algorithm.

A similar error appears in other experiments and the addition of the set of benign points into the optimization definitively solves it.

6.3 Weights of Benign Points for Classifier Training

In Section 5.3.2, we have discussed the possibilities of weighting the benign points for the training of classifiers. This section illustrates the experimental differences between these two cases.

We have performed an experiment with one-dimensional utility function with two local maxima and with the third dataset and the decision tree

classifier with maximum depth 2. The first run was with the weights equal to 1 for all points. The second experiment was with weights equal to $\frac{1}{n}$ for all points, where n is the number of points.

The first run with the weights of 1 has converged to the value 0.83237, and the second run has ended with the value 0,79168, which corresponds to the optimal value gained from the specialized discretization. The difference between these two runs occurs at the moment, when the attacker plays precisely the same point as the benign one, with the highest utility. In this situation, the classifier prefers to misclassify the point with the lower weight. Since the weight of the benign point is 1 and the point has utility 0.83237, it is preferred not to misclassify the adversarial point. Thus, the classifier misclassifies the adversarial one, which remains the best response strategy. The value of the game, in this case, is equal to the utility of the benign point. In the second run, the weight of the benign point is lower than its utility. Therefore, the algorithm adds to the restricted game the classifier, which misclassifies the benign point. It is played with a small probability not to violate the false-positive constraint. This strategy is the final one.

From this experiment, we can observe that the classifier does not prefer the misclassification of benign points when the weights are high. Thus, the weights equal to 1 can lead to higher final values. On the other hand, when the weights are too small, the classifier ignores the benign points, and the new classifier has a significant false-positive rate. Thus Double Oracle also ends with a high value. We can observe that the result also depends on the utilities of the benign points. Therefore, the optimal weights might depend on it.

We have used in all the experiments weights equal to 1, despite the worse results in the previous experiment. However, classifiers trained with these weights have a lower false-positive rate, which can cause an error during the convergence.

6.4 Decision Tree

In this section, we will describe experiments with the decision tree classifier. All the experiments have used the simultaneous best-response computation in the main loop of Double Oracle, discretization with local gradient optimization as the attacker's best-response algorithm, and weights of benign points equal to 1. The defender's best response is the decision tree, which has limited the maximal depth. We have tested the algorithm with different depths of the tree, with all three utility functions, and with the different number of dimensions of attacker's space. The results are shown in Tables 6.1, 6.2, and 6.3. For comparison, the optimal values are in Table 5.2.

We can observe that the experiments have converged close to the optimal value when the depth of the decision tree is large enough. With the increasing complexity of the utility function and with an increasing number of dimensions, the maximal depth of the decision tree necessary to converge close to the optimal value grows. Similar is the growth of the number of iterations needed

| | 1 | 2 | 3 |
|---|----|-----|-----|
| 1 | 41 | 490 | 302 |
| 2 | 41 | 149 | 177 |
| 3 | 41 | 157 | 248 |
| 4 | 41 | 425 | 290 |
| 5 | 41 | 227 | 401 |
| 6 | 41 | 264 | 432 |
| 7 | 41 | 283 | 416 |
| 8 | 41 | 294 | 396 |
| 9 | 41 | 465 | 390 |

(a) : Number of iterations

| | 1 | 2 | 3 |
|---|-------|-------|-------|
| 1 | 1.396 | 2.928 | 3.163 |
| 2 | 1.396 | 2.456 | 3.191 |
| 3 | 1.396 | 2.664 | 2.370 |
| 4 | 1.396 | 2.003 | 1.993 |
| 5 | 1.396 | 2.003 | 1.881 |
| 6 | 1.396 | 2.003 | 1.842 |
| 7 | 1.396 | 2.003 | 1.842 |
| 8 | 1.396 | 2.003 | 1.842 |
| 9 | 1.396 | 2.003 | 1.842 |

(b) : Final value of the game

Table 6.1: Results of experiments with the decision tree and the linear utility function. The columns are dimensions, the rows correspond to the maximal depth.

| | 1 | 2 | 3 |
|---|----|-----|-----|
| 1 | 2 | 9 | 10 |
| 2 | 50 | 167 | 463 |
| 3 | 50 | 273 | 407 |
| 4 | 50 | 257 | 497 |
| 5 | 50 | 197 | 422 |
| 6 | 50 | 339 | 409 |
| 7 | 50 | 308 | 557 |
| 8 | 50 | 536 | 792 |
| 9 | 50 | 362 | 833 |

(a) : Number of iterations

| | 1 | 2 | 3 |
|---|-------|-------|-------|
| 1 | 9.799 | 9.699 | 9.599 |
| 2 | 7.119 | 7.204 | 7.254 |
| 3 | 7.119 | 6.891 | 6.824 |
| 4 | 7.119 | 6.645 | 6.699 |
| 5 | 7.119 | 6.634 | 6.648 |
| 6 | 7.119 | 6.371 | 6.810 |
| 7 | 7.119 | 6.224 | 6.591 |
| 8 | 7.119 | 6.224 | 6.236 |
| 9 | 7.119 | 6.224 | 6.111 |

(b) : Final value of the game

Table 6.2: Results of experiments with the decision tree and the utility function with one maximum. The columns are dimensions, the rows correspond to the maximal depth.

| | 1 | 2 | 3 |
|---|-----|-----|-----|
| 1 | 5 | 41 | 38 |
| 2 | 133 | 11 | 18 |
| 3 | 133 | 68 | 52 |
| 4 | 133 | 250 | 56 |
| 5 | 133 | 281 | 65 |
| 6 | 133 | 367 | 138 |
| 7 | 133 | 365 | 133 |
| 8 | 133 | 447 | N/A |
| 9 | 133 | 436 | N/A |

(a) : Number of iterations

| | 1 | 2 | 3 |
|---|-------|-------|-------|
| 1 | 5.892 | 9.891 | 9.879 |
| 2 | 0.832 | 9.878 | 9.868 |
| 3 | 0.832 | 9.332 | 9.825 |
| 4 | 0.832 | 4.126 | 9.734 |
| 5 | 0.832 | 3.335 | 9.571 |
| 6 | 0.832 | 1.403 | 8.819 |
| 7 | 0.832 | 1.956 | 8.692 |
| 8 | 0.832 | 1.191 | N/A |
| 9 | 0.832 | 1.301 | N/A |

(b) : Final value of the game

Table 6.3: Results of experiments with the decision tree and the utility function with two maxima. The columns are dimensions, the rows correspond to the maximal depth.

for the convergence. We can see some deviations from this rule. For example, in the sixth and seventh row of Table 6.3. It is caused by the randomness of the selection of the dimension for a cut during the decision tree building. Thus, the computations differ even in the situations, where it is not necessarily the maximum depth of the tree. In the experiment with the utility with two maxima, two-dimensional attacker’s space and maximal depth 7, the random selection of the dimension causes the earlier usage of trees, which misclassifies some benign points, and the algorithm ends before reaching the optimal value. The cells filled with N/A correspond to the runs when the algorithm has not ended in the limit time, 24 hours.

A bottleneck of the algorithm is the optimization method for BR of the attacker. With the growing dimension, the memory needed for the discretization and the time complexity of the optimization algorithm grow exponentially. We demonstrate the time complexity on experiments with linear utility and with the maximal depth of the decision tree, equal to 5. The results can be seen in Figure 6.4. The calculation times are averaged over five runs, and the standard deviations are always smaller than 5 %.

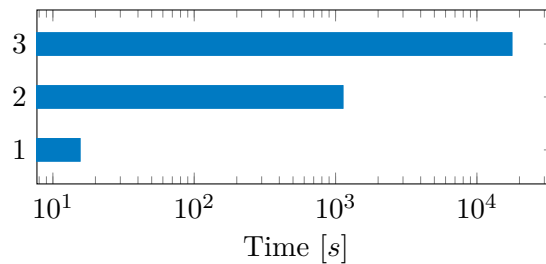


Figure 6.4: Time duration of the BR algorithm, depending on the dimension of the attacker’s space

The progress of convergence considerably varies among all three utility functions. We have plotted utility values of adversarial points, which have been added to the restricted game as the attacker’s best response. These values give us a curve of convergence of Double Oracle during the iterations.

The typical course of the convergence of Double Oracle with the linear utility looks like on Figure 6.5, where the convergence of experiment with the decision tree with maximal depth 6 and three-dimensional utility function is plotted. The y-axis corresponds to the utility of added adversarial point, and the x-axis corresponds to the iteration, in which the point was added. The green line in the graph corresponds to the value given by the smart discretization.

The oscillation at the beginning of the graph corresponds to the alternation of reactions to the newly added linear classifier and reactions to the mixture of previously added linear classifiers. Since the defender adds a classifier, which correctly classifies all previously added points, there usually is an attacker’s reaction with high utility. Thus, the optimal strategy in the next step mixes up between decision trees, which cut a different dimension. Optimal adversarial point reacting to the new strategy lies on the crossing of the two cuts and

has a lower utility than the previous best response.

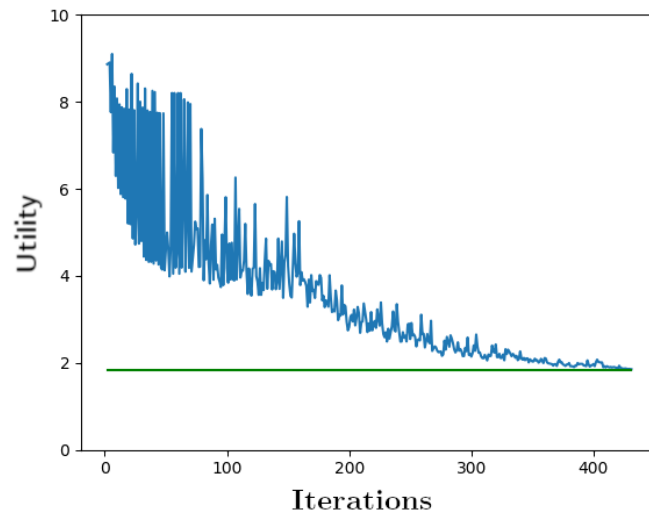


Figure 6.5: Convergence of DO with the three-dimensional linear utility and DT with max depth 6

The typical convergence course of the algorithm with the utility function with one maximum is plotted in Figure 6.6. The curve consists of three parts. In the first part, the utility is decreasing slowly. The points are separated by a shallow tree. In our example, the data are separated by a tree with depth 2. The breakpoint between the first and the second phase is the iteration of 50. In the second phase, the utility falls quickly. There are the spaces between the point clusters, which has a significant utility, cut out. The last part of the run starts around the iteration of 180. The decreasing slows down, and the defender generates deeper trees to cut a smaller space between the points.

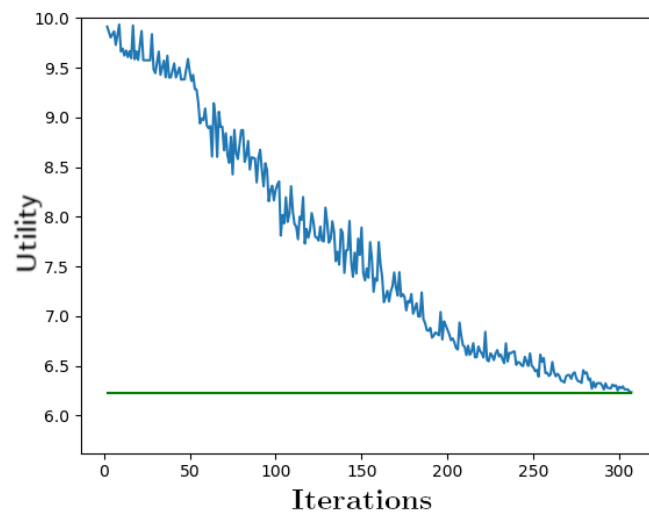
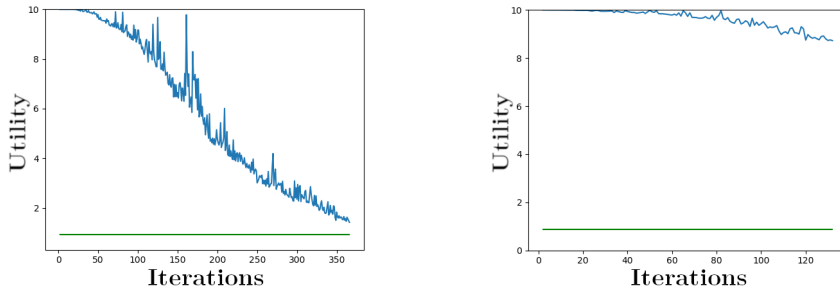


Figure 6.6: Convergence of DO with the two-dimensional utility with one maximum and DT with max depth 7

The hardest task for the decision tree is the utility function with two maxima. The benign points in these settings lie around the diagonal, which is complicated to separate by cuts in axes of individual dimensions. Thus, the maximal depth needed for optimal separation rapidly grows, and we do not reach it even in two dimensions. In Figure 6.7, the typical convergence of Double Oracle with the third utility function is displayed. We can see three similar phases as with the utility with one maximum, but the transitions are in this case smoother, and the first part of the curve is more significant than in the previous case. The courses, which do not converge near the optimal value, have progress looking like the first part of the typical run.



(a) : The two-dimensional utility and DT with max depth 6

(b) : The three-dimensional utility and DT with max depth 7

Figure 6.7: Convergence of DO with the utility with two maxima

6.4.1 Unlimited Decision Tree

We have run multiple experiments with different learning conditions. In the first one, we are increasing the depth of the tree, until the utility of it is higher than the utility of all classifiers added in the restricted game. In the other setting, we are increasing the depth of the tree, until the weighted change of misclassified points is higher than 0.1.

The version, where the first better classifier is added, has failed almost every time. We can demonstrate it on the experiment with the one-dimensional utility with one maximum. The attacker adds a point in between two clusters of benign points. The defender finds a decision tree, which separates the adversarial point from one of the clusters. The second cluster is misclassified. Since the tree has the utility 0, it is added to the restricted game. However, it has a high false-positive rate so it can be played with a small probability. The attacker's best response stays the same, and the algorithm ends with utility almost 10.

When we change the depth of the tree until the negligible improvement of the classification (smaller than 0.1), the algorithm always converges to the optimum. In this case, the tree does not misclassify a benign point, unless the deeper tree misclassifies it also. Therefore, the algorithm has converged in all runs to the optimal value. The curve of convergence has a similar shape as the curve of convergence of DO with decision tree with the smallest maximal depth, which ended near the optimal value. This allows us to find the best classifier with the smallest possible depths of the tree.

6.5 Support Vector Machine

This section describes experiments with Support Vector Machine classifier. All the experiments have used the simultaneous best-response computation in the main loop of Double Oracle, discretization with local gradient optimization as an attacker's best-response algorithm, and weights of benign points equal to 1. The defender's best response is SVM classifier with the polynomial kernel. We have tested the algorithm with different degrees of the polynomial kernel, with all three utility functions, and with a different number of dimensions of the attacker's space. The results are shown in Tables 6.4, 6.5, and 6.6. For comparison, the optimal values are in Table 5.2.

| | 1 | 2 | 3 |
|---|----|-----|-----|
| 1 | 22 | 75 | 118 |
| 2 | 30 | 95 | 225 |
| 3 | 36 | 119 | 358 |
| 4 | 43 | 175 | 342 |
| 5 | 50 | 141 | 408 |
| 6 | 54 | 145 | 424 |
| 7 | 62 | 175 | 446 |
| 8 | 66 | 183 | N/A |
| 9 | 72 | 195 | N/A |

(a) : Number of iterations

| | 1 | 2 | 3 |
|---|-------|-------|--------|
| 1 | 1.398 | 2.087 | 1.8939 |
| 2 | 1.402 | 2.037 | 1.8638 |
| 3 | 1.399 | 2.007 | 1.8421 |
| 4 | 1.396 | 2.003 | 1.8421 |
| 5 | 1.396 | 2.003 | 1.8449 |
| 6 | 1.396 | 2.003 | 1.8421 |
| 7 | 1.396 | 2.003 | 1.8421 |
| 8 | 1.396 | 2.003 | N/A |
| 9 | 1.396 | 2.003 | N/A |

(b) : Final value of the game

Table 6.4: Results of experiments with the SVM and the linear utility function. The columns are dimensions, the rows correspond to the degree of kernel.

| | 1 | 2 |
|---|----|-----|
| 1 | 2 | 2 |
| 2 | 14 | 107 |
| 3 | 22 | 113 |
| 4 | 28 | 136 |
| 5 | 31 | 116 |
| 6 | 33 | N/A |
| 7 | 36 | N/A |
| 8 | 37 | N/A |
| 9 | 39 | N/A |

(a) : Number of iterations

| | 1 | 2 |
|---|-------|-------|
| 1 | 9.799 | 9.737 |
| 2 | 7.186 | 6.318 |
| 3 | 7.123 | 6.378 |
| 4 | 7.122 | 6.330 |
| 5 | 7.126 | 6.325 |
| 6 | 7.910 | N/A |
| 7 | 7.515 | N/A |
| 8 | 7.515 | N/A |
| 9 | 7.119 | N/A |

(b) : Final value of the game

Table 6.5: Results of experiments with the SVM and the utility function with one maximum. The columns are dimensions, the rows correspond to the degree of kernel.

In the results, we can observe the expected trends, similar to trends with the decision tree. With a growing degree of the kernel, the number of iterations grows, and the final value is approaching the optimal value calculated by the smart discretization. However, the SVM classifier is sensitive to positions

| | 1 | 2 | 3 |
|---|-----|-----|-----|
| 1 | 5 | 5 | 5 |
| 2 | 122 | 104 | 181 |
| 3 | 161 | 121 | 324 |
| 4 | 112 | 161 | 298 |
| 5 | 130 | 139 | 253 |
| 6 | 66 | 212 | 373 |
| 7 | 70 | N/A | N/A |
| 8 | 86 | N/A | N/A |
| 9 | 85 | N/A | N/A |

(a) : Number of iterations

| | 1 | 2 | 3 |
|---|-------|-------|-------|
| 1 | 5.892 | 6.694 | 6.739 |
| 2 | 0.819 | 0.921 | 0.751 |
| 3 | 0.791 | 0.921 | 0.868 |
| 4 | 0.791 | 0.921 | 1.169 |
| 5 | 0.791 | 0.921 | 1.297 |
| 6 | 0.791 | 0.926 | 1.218 |
| 7 | 0.791 | N/A | N/A |
| 8 | 0.791 | N/A | N/A |
| 9 | 0.791 | N/A | N/A |

(b) : Final value of the game

Table 6.6: Results of experiments with the SVM and the utility function with two maxima. The columns are dimensions, the rows correspond to the degree of kernel.

and weights of the benign data, because there is a component affecting the structure of the classifier in training. Thus, the defender can prefer to misclassify points, even when they are separable. It can lead to the "failure" of the classifier, and the value does not reach the expected value.

The progress of the convergence during the time is similar for the first and the third utility function. We can see it in Figure 6.8. SVM classifier crops the extremes of the utility function quickly and then slightly improves the classifiers to get the optimum.

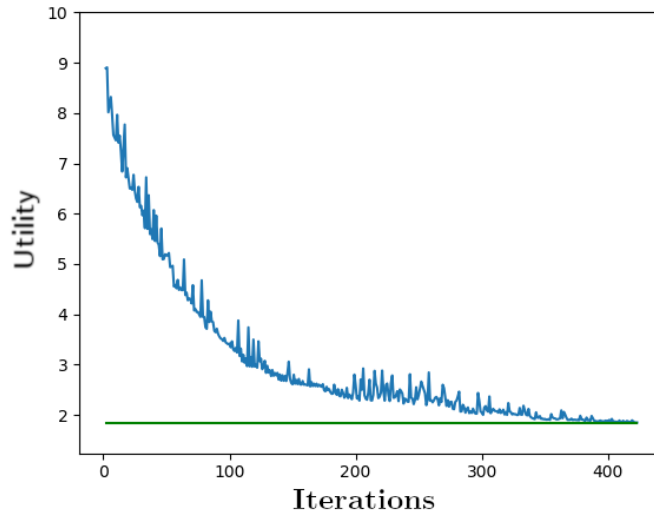


Figure 6.8: Convergence of DO with the three-dimensional linear utility and SVM with polynomial kernel with degree 6

The interesting difference appears in runs with the utility function with two maxima and with SVM classifier with a kernel with a degree higher than needed. Such a course is shown in Figure 6.9. In these cases, the classifier overfits the data, which causes the oscillation at the beginning, and the convergence is slowed down.

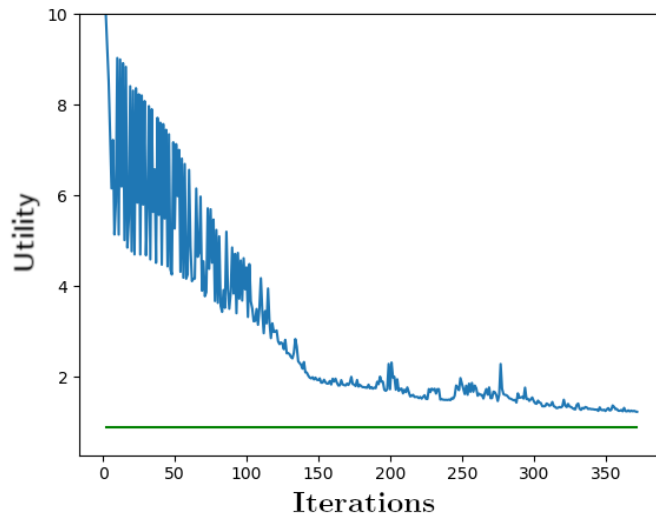


Figure 6.9: Convergence of DO with the three-dimensional utility with two maxima and SVM with polynomial kernel with degree 6

Experimental results show that the problems with the utility function with one maximum are the hardest for SVM classifier. The typical run is plotted in Figure 6.10. The drop at the beginning of the run is much smaller than in the previous cases. In the situation, with one local maximum, the classifier is forced to cut out the space between the clusters of benign points generated in corners. Therefore, the curve linearly decreases.

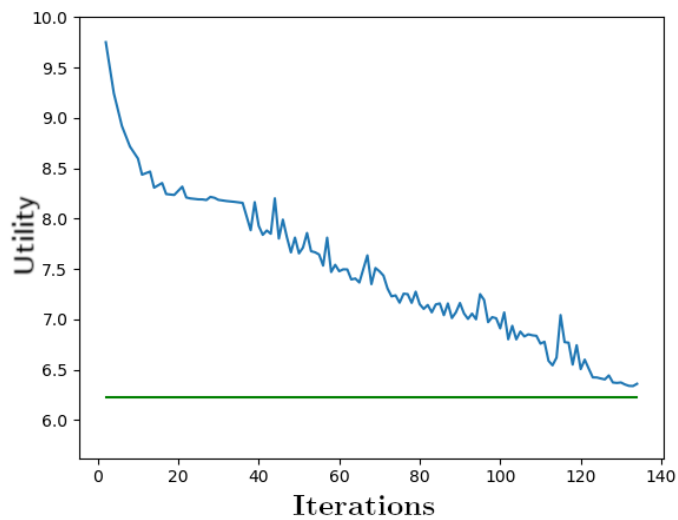


Figure 6.10: Convergence of DO with the two-dimensional utility with one maximum and SVM with polynomial kernel with degree 4

The training of the SVM classifier, cutting out the space between the points in corners, takes longer than in experiments with other utility functions. For example, we can compare training times of SVM with a polynomial kernel with degree 5, during experiments with two-dimensional utility function with one

maximum and with two maxima. The training times are shown in Figure 6.11. In the problem with two local maxima, the defender needs to train a classifier separating the benign points and the maxima, which can be done with two straight lines around the benign points. By contrast, the cutting of the space between point clusters needs more complex decision boundary.

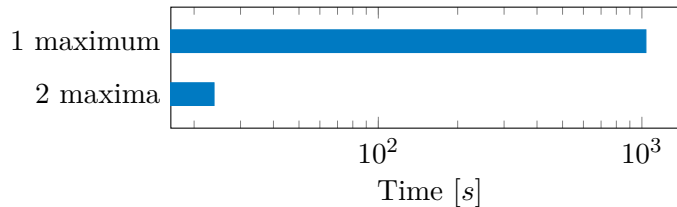


Figure 6.11: Time duration of training of SVM in seconds, depending on the utility function

6.6 Neural Network

Double Oracle with the neural network has multiple parameters to be set. The first parameter is the number of neurons in the hidden layer. It is the only used parameter, which affects the strength of the classifier. The other two parameters are the number of epochs and the number of iterations. The training of the neural network is done in loops. In each iteration, the neural network learns for epochs, and then the algorithm checks the actual utility of the classifier. We run experiments with setting 1000 epochs and 100 iterations. As the initialization of weights in NN, the network from the previous iteration of Double Oracle is used.

In the first experiments, we have added the first neural network, which has by 0.5 % better utility than all classifiers in the restricted game. These experiments have two possible scenarios. In the first one, the algorithm quickly fails because it finds a classifier, which has a high false-positive rate and classifies all adversarial points correctly. Thus, the utility of such classifier is 0, but it cannot be played with a significant probability. The probability of this failure rapidly grows with random initialization of weights in each iteration of the Double-Oracle algorithm.

The convergence course in the second scenario is plotted in Figure 6.12. It quickly drops near to the optimal value, but the decreasing of the value quickly slows down. In this part of the convergence, the training is very sensitive to the weights of benign data. When the weights are lower, the curve approaches to the optimum at the shorter distance, but there is a higher risk of misclassification of a lot of benign points, which leads to the absolute failure of Double Oracle.

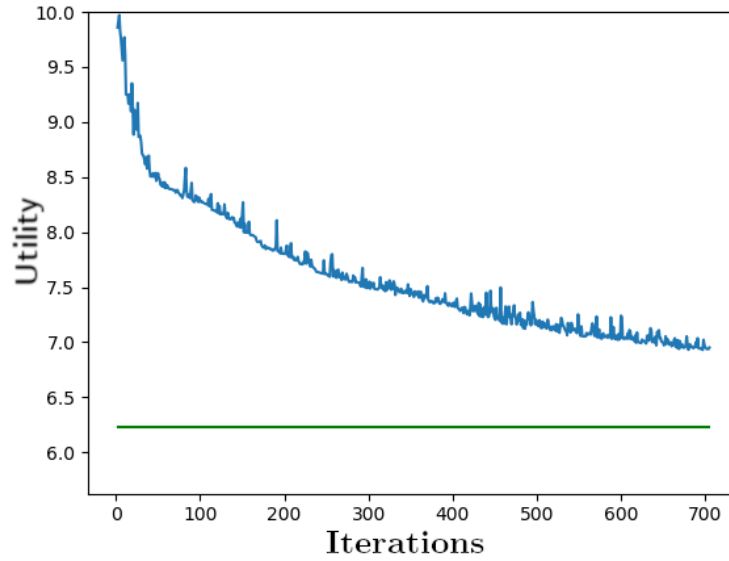


Figure 6.12: Convergence of DO with the two-dimensional linear utility and NN with 10 neurons in hidden layer, adding first better

The second batch of experiments has continued in the training of the neural network, until the loss drops under the value of 0.01. This modification eliminates the failure caused by the misclassification of benign points. However, we do not observe any other significant difference between the courses of convergence in these experiments and in the previous ones.

The exemplary results are shown in Tables 6.7, 6.8, and 6.9. In these experiments, we have used the second training condition with the threshold on the change of the loss equal to 0.01. For comparison, the optimal values are in Table 5.2.

| | 1 | 2 | 3 |
|----|-----|-----|-----|
| 1 | 2 | 2 | 3 |
| 2 | 7 | 9 | 9 |
| 4 | 36 | 70 | 152 |
| 7 | 67 | 182 | 256 |
| 9 | 72 | 108 | 319 |
| 10 | 85 | 230 | N/A |
| 20 | 105 | 657 | N/A |

(a) : Number of iterations

| | 1 | 2 | 3 |
|----|-------|-------|-------|
| 1 | 9.899 | 9.872 | 9.158 |
| 2 | 1.720 | 4.454 | 8.937 |
| 4 | 1.862 | 3.404 | 3.943 |
| 7 | 1.600 | 2.878 | 3.547 |
| 9 | 1.558 | 2.992 | 3.536 |
| 10 | 1.506 | 2.496 | N/A |
| 20 | 1.503 | 2.376 | N/A |

(b) : Final value of the game

Table 6.7: Results of experiments with the NN and the linear utility function. The columns are dimensions, the rows correspond to the number of neurons in hidden layer.

In Double Oracle with a neural network, the time necessary for training nearly competes with the time complexity of the attacker's best response.

| | 1 | 2 | 3 |
|----|----|-----|-----|
| 1 | 3 | 1 | 1 |
| 2 | 5 | 2 | 1 |
| 4 | 12 | 79 | 97 |
| 7 | 9 | 286 | 240 |
| 9 | 14 | 551 | N/A |
| 10 | 18 | 738 | N/A |
| 20 | 27 | N/A | N/A |

(a) : Number of iterations

| | 1 | 2 | 3 |
|----|-------|-------|-------|
| 1 | 9.866 | 10.00 | 10.00 |
| 2 | 9.873 | 9.998 | 10.00 |
| 4 | 9.788 | 9.309 | 9.867 |
| 7 | 9.799 | 8.344 | 9.741 |
| 9 | 8.876 | 7.135 | N/A |
| 10 | 8.769 | 6.835 | N/A |
| 20 | 7.644 | N/A | N/A |

(b) : Final value of the game

Table 6.8: Results of experiments with the NN and the utility function with one maximum. The columns are dimensions, the rows correspond to the number of neurons in hidden layer.

| | 1 | 2 | 3 |
|----|----|-----|-----|
| 1 | 4 | 2 | 3 |
| 2 | 5 | 4 | 1 |
| 4 | 12 | 41 | 83 |
| 7 | 17 | 293 | N/A |
| 9 | 34 | 416 | N/A |
| 10 | 29 | N/A | N/A |
| 20 | 53 | N/A | N/A |

(a) : Number of iterations

| | 1 | 2 | 3 |
|----|-------|-------|-------|
| 1 | 9.900 | 10.00 | 10.00 |
| 2 | 6.636 | 9.891 | 10.00 |
| 4 | 5.014 | 7.505 | 8.610 |
| 7 | 4.950 | 5.420 | N/A |
| 9 | 2.846 | 4.598 | N/A |
| 10 | 3.195 | N/A | N/A |
| 20 | 1.765 | N/A | N/A |

(b) : Final value of the game

Table 6.9: Results of experiments with the NN and the utility function with two maxima. The columns are dimensions, the rows correspond to the number of neurons in hidden layer.


6.7 Final Observations

At the end of the chapter with experiments, we would like to introduce two general observations about the adversarial classification games.

The adversarial classification problem seems to be in close association with the Stackelberg structure of the problem. In the majority of experiments, the attacker has the support smaller than 10. Moreover, in almost half of the experiments, the attacker plays a pure point. It is because the defender makes a commitment to use the classifiers, which separates the benign data from the maxima of the utility function, and the attacker finds the best reaction to it. For these reasons, it would be interesting to look at the problem from the view of Stackelberg equilibrium.

The second interesting observation is the size of the support of the defender. He usually mixes over tens or hundreds of classifiers. It leads to two ideas. The defender in practical cases would like to use as few classifiers as possible because each classification in the application costs something. Therefore, we have to add the number of classifiers into the objective of the linear program in order to minimize it.

With a deeper look, we can find out, that many of the classifiers are dominated by others. Since none of them classifies the adversarial points correctly, it does not matter, which one has a nonzero probability. Thus, we can modify the defender's strategy to minimize the support, without a change of the expected value. Moreover, we can filter dominated classifiers during Double Oracle, which would speed up the run of the algorithm. We would have to be aware of the false-positive rate and not only of the game matrix.



Chapter 7

Conclusions

In this work, we have decided to study the usage of the Double-Oracle algorithm for finding an approximation of a Nash equilibrium in infinite games, with motivation to find a robust solution of the adversarial classification problem. Therefore, we have summarized the definitions from game theory, which are related to problems with the games with infinite strategy space. In the second chapter, we have introduced the solution concept Nash equilibrium and a few algorithms, how to find it in finite normal-form games. Especially, we have focused on Double Oracle, which can be generalized to infinite games.

Next part of the work is focused on the problem of adversarial classification from the game theoretic view. We have formalized the problem as a general-sum game, and we have shown pitfalls of using Double Oracle and of finding the Nash equilibrium in it. Thus, we have proposed three alternatives, how to redefine it as a zero-sum or an almost zero-sum game. From these definitions, we have selected the adversarial classification game with hard false-positive constraint in expectation for experiments. For this representation, we have found a linear program, which computes an exact value of the game, but it overfits the data.

Based on this theoretical background, we have prepared a framework for simple usage of Double Oracle in adversarial classification problems. The framework is easily modifiable by exchanging the classifier or the utility function of the attacker. We have implemented two options for calculation of the attacker's best response and prepared the decision tree, the SVM, and the neural network classifier for its usage in the framework. Finally, we have prepared three different utility functions and datasets of benign points parametric in the number of dimensions.

The main part of the work consists of the results of the experiments on our framework, which map the usability of Double Oracle in infinite games, specifically, in adversarial classification problem games with hard constraint in expectation. Double Oracle converges near to the value gained by the exact algorithm in all cases, where the classifier has sufficient strength. Unfortunately, there is a bottleneck in the form of the attacker's BR algorithm. In large measure, it is probably caused by the implementation in Python. Therefore, Double Oracle solves problems with the dimension of the attacker's space smaller than 4.

On the other hand, in some experiments, the training of a classifier takes a significant part of the computation time, especially in experiments with neural network, and in experiments with SVM and the utility function with one maximum.

We have evaluated experiments, which have shown the difference between the alternating and the simultaneous computation of the best response in the main loop of Double Oracle. Based on these experiments, we decided to use the standard simultaneous form, which is robust and works correctly in all cases.

We have also performed experiments, which examine the weighting of benign points for the training. Different weights have an impact on the fact, how well the trained classifier approximates to the defender's best response. We have found out that with lower weights, Double Oracle can converge to the optimum, but there is a higher risk of absolute failure.

The next experiments have been focused on the question, if it is better to use the first classifier with better utility than all others in the restricted game, or if it is better to train the classifier longer. Since the expected utility does not reflect the misclassification of the benign points, it is beneficial to train the classifier longer, so it gets a lower false-positive rate.

Eventually, we have discussed the structure of the strategy, which has been usually found. There is interesting the Stackelberg character of the problem, which is noticeable even in an approximation of a Nash equilibrium, where the attacker mixes between only a few points. On the other hand, the support of the classifier is usually large, which can be in contrast with the need for a small and fast solution.

7.1 Future Work

This work opens many ways in the field of usage of Double Oracle for solving infinite games and in the field of game theoretic approaches to the problems of adversarial classification.

First steps could lead to the improvement of scalability of the algorithm. The optimization algorithm used for finding the best response of the attacker could be implemented in another programming language, and the implementation could be parallelized to be run on GPU. The same way, we could use the more specialized implementations of classifiers.

The next step could be an improvement of the main loop of Double Oracle. For example, we could remove old dominated classifiers during the run.

Since we have an algorithm, which calculates the exact value of a Nash equilibrium, the main advantage of Double Oracle is a generalization of the benign points dataset. Therefore, it would be interesting to perform experiments evaluating this statement and run Double Oracle with classifiers using the cross-validation. It is prepared in the framework.

We have seen, the distinctness of Stackelberg structure in the problem. Therefore, it could be interesting to find Stackelberg equilibrium in the adversarial classification game and compare it to the found Nash equilibrium.

Double Oracle applied on adversarial classification problem can be seen as a kind of Boosting. In each iteration, there are added new classifiers, and points in the space are reweighted. Probably, there can be found a relationship between these two approaches.

Appendix A

Bibliography

- [1] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *CoRR*, vol. abs/1412.6572, 2015.
- [2] R. Axelrod, “Effective choice in the prisoner’s dilemma,” *Journal of Conflict Resolution – J CONFLICT RESOLUTION*, vol. 24, pp. 3–25, 01 1980.
- [3] B. Bosanský, C. Kiekintveld, V. Lisý, and M. Pechoucek, “An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information,” *Journal of Artificial Intelligence Research*, vol. 51, pp. 829–866, 2014.
- [4] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, “Flexible, high performance convolutional neural networks for image classification.,” *International Joint Conference on Artificial Intelligence IJCAI-2011*, pp. 1237–1242, 07 2011.
- [5] D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, “A committee of neural networks for traffic sign classification,” *The 2011 International Joint Conference on Neural Networks*, pp. 1918–1921, 2011.
- [6] D. Połap and M. Woźniak, “Voice recognition by neuro-heuristic method,” *Tsinghua Science and Technology*, vol. 24, pp. 9–17, Feb 2019.
- [7] M. Rozenwald, E. Khrameeva, G. Sapunov, and M. Gelfand, “Prediction of 3d chromatin structure using recurrent neural networks,” in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 2488–2488, Dec 2018.
- [8] M. Yeo, Y. Koo, Y. Yoon, T. Hwang, J. Ryu, J. Song, and C. Park, “Flow-based malware detection using convolutional neural network,” in *2018 International Conference on Information Networking (ICOIN)*, pp. 910–913, Jan 2018.
- [9] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, “Malware detection with deep neural network using process behavior,” in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 577–582, June 2016.

- [10] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. 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, (New York, NY, USA), pp. 506–519, ACM, 2017.
- [11] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 372–387, March 2016.
- [12] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, “Evasion attacks against machine learning at test time,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 387–402, 2013.
- [13] D. Lowd and C. Meek, “Good word attacks on statistical spam filters.,” in *CEAS*, 2005.
- [14] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, “Adversarial examples for malware detection,” in *Computer Security – ESORICS 2017* (S. N. Foley, D. Gollmann, and E. Sneekenes, eds.), (Cham), pp. 62–79, Springer International Publishing, 2017.
- [15] J. W. Stokes, D. Wang, M. Marinescu, M. Marino, and B. Bussone, “Attack and defense of dynamic analysis-based, adversarial neural malware detection models,” in *IEEE Military Communications Conference (MILCOM)*, pp. 1–8, Oct 2018.
- [16] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow, “Thermometer encoding: One hot way to resist adversarial examples,” in *International Conference on Learning Representations*, 2018.
- [17] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” *CoRR*, vol. abs/1611.01236, 2016.
- [18] P. Samangouei, M. Kabkab, and R. Chellappa, “Defense-GAN: Protecting classifiers against adversarial attacks using generative models,” in *International Conference on Learning Representations*, 2018.
- [19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *International Conference on Learning Representations*, 2018.
- [20] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, “Pixeldefend: Leveraging generative models to understand and defend against adversarial examples,” in *International Conference on Learning Representations*, 2018.

- [21] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, “Adversarial attacks and defences: A survey,” *CoRR*, vol. abs/1810.00069, 2018.
- [22] D. Balduzzi, S. Racanière, J. Martens, J. N. Foerster, K. Tuyls, and T. Graepel, “The mechanics of n-player differentiable games,” in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 363–372, 2018.
- [23] A. Athalye, N. Carlini, and D. A. Wagner, “Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples,” in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 274–283, 2018.
- [24] M. Brückner, C. Kanzow, and T. Scheffer, “Static prediction games for adversarial learning problems,” *The Journal of Machine Learning Research*, vol. 13, pp. 2617–2654, 09 2012.
- [25] M. Brückner and T. Scheffer, “Stackelberg games for adversarial prediction problems,” pp. 547–555, 08 2011.
- [26] S. R. Bulò, B. Biggio, I. Pillai, M. Pelillo, and F. Roli, “Randomized prediction games for adversarial machine learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 2466–2478, Nov 2017.
- [27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.
- [28] H. B. McMahan, G. J. Gordon, and A. Blum, “Planning in the presence of cost functions controlled by an adversary,” in *ICML* (T. Fawcett and N. Mishra, eds.), pp. 536–543, AAAI Press, 2003.
- [29] M. Lanctot, V. F. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, “A unified game-theoretic approach to multiagent reinforcement learning,” *Advances in Neural Information Processing Systems*, pp. 4190–4203, 2017.
- [30] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. New York, NY, USA: Cambridge University Press, 2008.
- [31] N. D. Stein, P. A. Parrilo, and A. Ozdaglar, “Characterization and computation of correlated equilibria in infinite games,” in *2007 46th IEEE Conference on Decision and Control*, pp. 759–764, Dec 2007.

- [32] M. Dresher, S. Karlin, and L. S. Shapley, “Polynomial games,” *In Contributions to the Theory of Games, Annals of Mathematics Studies*, vol. 24, pp. 161–180, 1950.
- [33] J. Rehbeck, “Note on unique nash equilibrium in continuous games,” *Games and Economic Behavior*, vol. 110, pp. 216–225, 2018.
- [34] M. Jamil and X.-S. Yang, “A literature survey of benchmark functions for global optimization problems,” *IJMNO*, vol. 4, pp. 150–194, 2013.
- [35] J. Nash, “Non-cooperative games,” *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.
- [36] L. E. J. Brouwer, “Über abbildung von mannigfaltigkeiten,” *Mathematische Annalen*, vol. 71, pp. 598–598, Dec 1912.
- [37] I. L. Glicksberg, “A further generalization of the kakutani fixed point theorem, with application to nash equilibrium points,” *Proceedings of the American Mathematical Society*, vol. 3, no. 1, pp. 170–174, 1952.
- [38] N. D. Stein, A. Ozdaglar, and P. A. Parrilo, “Separable and low-rank continuous games,” *International Journal of Games Theory*, vol. 37, pp. 475–504, 12 2008.
- [39] J. v. Neumann, “Zur theorie der gesellschaftsspiele,” *Mathematische Annalen*, vol. 100, pp. 295–320, Dec 1928.
- [40] B. von Stengel, “Computing equilibria for two-person games,” in *Handbook of Game Theory with Economic Applications* (R. Aumann and S. Hart, eds.), vol. 3, ch. 45, pp. 1723–1759, Elsevier, 1 ed., 2002.
- [41] D. P. Dobkin and S. P. Reiss, “The complexity of linear programming,” *Theoretical Computer Science*, vol. 11, no. 1, pp. 1 – 18, 1980.
- [42] T. Sandholm, A. Gilpin, and V. Conitzer, “Mixed-integer programming methods for finding nash equilibria.,” vol. 2, pp. 495–501, 01 2005.
- [43] C. E. Lemke and J. T. Howson, “Equilibrium points of bimatrix games,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 2, pp. 413–423, 1964.
- [44] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, “The complexity of computing a nash equilibrium,” in *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing, STOC '06*, (New York, NY, USA), pp. 71–78, ACM, 2006.
- [45] P. A. Parrilo, “Polynomial games and sum of squares optimization,” in *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 2855–2860, Dec 2006.

- [46] R. J. Lipton, E. Markakis, and A. Mehta, “Playing large games using simple strategies,” in *Proceedings of the 4th ACM Conference on Electronic Commerce, EC '03*, (New York, NY, USA), pp. 36–41, ACM, 2003.
- [47] B. Bosansky, V. Lisy, M. Lanctot, J. Cermak, and M. Winands, “Algorithms for computing strategies in two-player simultaneous move games,” *Artificial Intelligence*, vol. 237, 04 2016.
- [48] M. Jain, D. Korzhyk, O. Vaněk, V. Conitzer, M. Pechoucek, and M. Tambe, “A double oracle algorithm for zero-sum security games on graphs,” in *AAMAS*, 2011.
- [49] H. von Stackelberg, D. Bazin, R. Hill, and L. Urch, *Market Structure and Equilibrium*. Springer Berlin Heidelberg, 2010.
- [50] G. Leitmann, “On generalized stackelberg strategies,” *Journal of Optimization Theory and Applications*, vol. 26, pp. 637–643, 12 1978.
- [51] D. J. Wales and J. P. K. Doye, “Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms,” *The Journal of Physical Chemistry A*, vol. 101, no. 28, pp. 5111–5116, 1997.
- [52] R. Byrd, P. Lu, J. Nocedal, and C. Zhu, “A limited memory algorithm for bound constrained optimization,” *SIAM Journal of Scientific Computing*, vol. 16, pp. 1190–1208, 9 1995.

Appendix B

Experiments on Discretization Algorithm

We have tested the scalability of the basic discretization algorithm for the approximation of a Nash equilibrium in infinite games from Chapter 4.2.

The algorithm was evaluated on randomly generated polynomial zero-sum game with maximum degree smaller than 10. The dimension of the strategy space of the first player is the scaled parameter, and the second player has a one-dimensional strategy space all the time.

The algorithm had 1000 samples in each dimension. The results are plotted in Figure B.1.

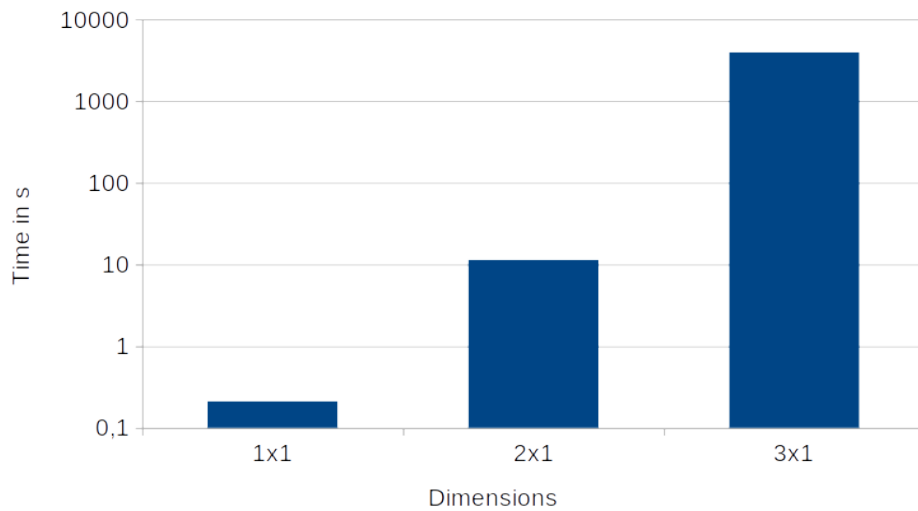


Figure B.1: Solution times of discretization algorithm for scaled dimension count in seconds (player 1 dimensions \times player 2 dimensions)

The discretization algorithm was implemented in Python 3.7.1 using NumPy 1.15.4, and linear programs were solved using Gurobi 7.5.2. All the experiments run on a computer with Intel(R) Core(TM) i7-7820HQ CPU @ 2.90GHz, 16GB RAM.

Appendix C

Framework Source Code

We have implemented a framework for simple usage of the Double-Oracle algorithm in adversarial classification problem in Python 3.6. The used libraries with versions are listed in Table C.1

| Library | version |
|-----------------------------|---------|
| Python ¹ : | 3.6.2 |
| NumPy ² : | 1.16.1 |
| SciPy ³ : | 1.1.0 |
| Gurobi ⁴ : | 7.5.2 |
| CVXOPT ⁵ : | 1.2.3 |
| scikit-learn ⁶ : | 0.19.1 |
| PyTorch ⁷ : | 0.3.0 |
| matplotlib ⁸ : | 3.0.3 |

Table C.1: The version of software used in the framework

The source code consists of three main parts: main loop, optimization for computation of the attacker’s best response and the defender’s best response classifier.

The main loop of Double Oracle is located in *DoubleOracle* class. There are multiple options adjustable with arguments of the init function. They influence the run of the algorithm. We can set the optimizer for solving LP, negative data weights, or the type of the main loop – alternating, simultaneous or mixing calculation of best responses of the players. Algorithms for calculation of best response are also the parameters of the function. Computation is started by calling the function *compute*.

The attacker’s best response is calculated by a class inheriting from *OptimizationInterface* abstract class. It has to call the parents constructor and

¹<https://www.python.org/>

²<https://www.numpy.org/>

³<https://www.scipy.org/>

⁴<http://www.gurobi.com/>

⁵<https://cvxopt.org/>

⁶<https://scikit-learn.org/>

⁷<https://pytorch.org/>

⁸<https://matplotlib.org/>

implement the *optimize* method, which returns the optimal point and its value. In the framework, there are prepared three options discussed in the work – Basin-Hopping algorithm, Basin-Hopping algorithm with discretization, and discretization with L-BFGS-B local optimization.

The defender’s best response classifier inherits from *ProbabilisticClassifier* abstract class. It has to override the *classify_by_one_classifier* method, which implements the prediction of the classifier, and the *update* method, which trains a new classifier and tries to add it to the game using the pre-implemented *expand_classifiers* method. When the classifier does not support the hard constraining the false-positive rate during training, it has to inherit from *NotAllowHardFP* class. Moreover, when the classifier can easily get the distance of a point from the decision boundary, it has to inherit from *ProbabilisticClassifierWithBoundary*, which allows transforming the distance into the probability of the class.

In the framework, there are prepared three classifiers: decision tree, Support Vector Machine, and neural network. Decision tree allows to restrict the maximal depth of the tree, or it can iteratively deepen the tree until there is an improvement of the classification higher than a threshold. SVM has an optional kernel and all other parameters of the classifier. The neural network has a predefined model with one hidden layer, but it can be exchanged in parameters. Similarly, we can exchange the loss function, optimizer, and all the parameters. The training of the neural network is executed in loops. In one loop it runs the backpropagation for epochs and then it checks the utility of the classifier. The number of epochs and the number of iterations can be set with parameters. There are two options for the end of the training. In the first option, the classifier adds the first classifier with better utility, and in the second option, the training continues, until the change of the loss is smaller than the threshold. The last parameter sets the initialization of the neural network. In the first option the weights are set randomly, and in the second one, the algorithm continues in training of the network from the previous iteration of Double Oracle.

Next, to the framework, there is also the smart discretization solver from Section 4.5.

We have prepared a `main.py` file, which enables a simple run of the framework from the console. All parameters of the script are described in the help:

Usage of the script:

```
python main.py function points fp_threshold algorithm *params
                [optimizer] [step] [weights]
```

```
function:      0 -> Linear utility
                1 -> Utility with one maximum
                2 -> Utility with two maxima
```

```
points:        a path to the *.numpy file with the benign points
```


Appendix D

CD Content

The enclosed CD contains following files and directories:

- **silhapro.pdf** - The text of this thesis
- **text_source** - The source code of the text
 - **appendices** - The appendices of the work
 - **chapters** - The chapters source code
 - **img** - The figures
 - **specification** - The specification of the thesis
- **data** - The datasets used for the experiments
 - *The generated datasets*
 - **generate_dataset.py** - The script for generation of datasets
- **framework** - The source code of the framework

The text source code is written in L^AT_EX using the template CTUstyle created by Petr Olšák¹.

For generation of new datasets, you can use the script `generate_dataset.py`.

Usage of the script:

```
python generate_dataset.py generator dimensions name
```

```
generator:      0 -> First dataset  
                1 -> Second dataset  
                2 -> Third dataset
```

```
dimensions:    the number of dimensions of the points
```

```
name:          name of the generated file
```

¹<http://petr.olsak.net/ctustyle.html>