

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Уфимский государственный нефтяной технический университет»  
Факультет автоматизации производственных процессов  
Кафедра «Вычислительная техника и инженерная кибернетика»

УДК 004.032.26

К ЗАЩИТЕ ДОПУЩЕНА

И. о. зав. кафедрой ВТИК, доц.,  
канд. физ.-мат. наук

\_\_\_\_\_ Д.М. Зарипов

\_\_\_\_\_  
(дата)

**COMPARISON OF THE RESULTS OF AN ARTIFICIAL NEURAL NETWORK  
WITH SEQUENTIAL TRUNCATION OF THE WEIGHT MATRIX**

Выпускная квалификационная работа  
(бакалаврская работа)  
по направлению подготовки 09.03.01 Информатика и вычислительная техника,  
профиль «Программное обеспечение средств вычислительной техники и  
автоматизированных систем»

0.200.00.00.00676ПЗ

Студент группы БПОи 16-01

К.И. Камалов

Руководитель, доц., канд. техн. наук

В.М. Гиниятуллин

Нормоконтролер

М.А. Салихова

Уфа 2020

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
"Уфимский государственный нефтяной технический университет"  
Кафедра Вычислительная техника и инженерная кибернетика  
(наименование кафедры)

## ЗАДАНИЕ на выполнение бакалаврской работы

Студент Камалов Камиль Илиатович группа БПОи-16-01  
(фамилия, имя, отчество полностью) (шифр)

Тема бакалаврской работы (БР) Comparison of the results of an artificial neural network with sequential truncation of the weight matrix

Дата утверждения темы БР на заседании кафедры ВТИК « 12 » марта 2020 г.,  
протокол № 6

Срок представления БР к защите «22» июня 2020 г.

Исходные данные к выполнению БР:

- 1) материалы, собранные студентом при прохождении преддипломной практики;
- 2) дополнительные данные:

Объем текстовой части БР: 80 листов (страниц) формата А4.

Перечень основных структурных элементов текстовой части БР:

1. Содержание ( 2 с.).
2. Реферат ( 1 с.).
3. Введение ( 2 с.).
4. Review of ANN, backgammon and development methodologies

Раздел, содержание которого определяется спецификой БР (литературный обзор, патентный анализ,

обзор законодательных и нормативных актов, характеристика объекта исследования и т.п.)

( 28 с.).

5. Основная часть, включающая разделы: \_\_\_\_\_  
Структура основной части определяется спецификой и тематикой БР

Design software for managing artificial neural network instances

( 22 с.).

6. Другие разделы Software implementation, testing and obtaining results

Разделы, содержание которых определяются спецификой БР (экономической, автоматизации,

( 6 с.).

безопасности и экологичности и др.)

7. Список использованных источников ( 28 наименов.). 8. Приложения: Appendix A (obligatory),  
Перечень приложений

Appendix B (recommended), Appendix C (recommended)

Объем и перечень иллюстрационно-графического материала 54 figures., 7 tables, presentation, video clip

Консультанты по разделам БР (с указанием относящихся к ним разделов)

Задание выдал:

Руководитель БР

В.М. Гиниятуллин

(подпись)

(И.О. Фамилия)

13.03.2020

(дата)

Задание получил:

Студент

К. И. Камалов

(подпись)

(И.О. Фамилия)

13.03.2020

(дата)

## **ABSTRACT**

Bachelor's thesis, 80 pages, 54 figures, 7 tables, 28 sources, 3 appendixes.

ARTIFICIAL NEURAL NETWORK, ARTIFICIAL NEURON, BLACK BOX, MACHINE LEARNING, LEARNING ERA, UNDERSTANDING, EVALUATION CRITERIA, BACKGAMMON, GAME RESULTS, WEIGHT MATRIX

The object of the study is the results of the ANN for playing backgammon at various eras of training.

In the process of researching the work, the subject area was examined, changes in the results of the ANN were studied with successive truncation of the matrix of weights, and the need for refactoring the ANN was substantiated.

The purpose of the work is to determine the criteria for evaluating the results of the ANN.

As a result of the study, a software product for playing backgammon was implemented with the ability to configure the ANN to receive prompts and play against it.

The novelty of the work is to determine the relationship between the inputs and outputs of the ANN by sequentially truncating its weight matrix.

The practical significance of the results of the work lies in the possibility of introducing understanding into the work of the ANN.

## TABLE OF CONTENTS

Nomenclature.....	6
Introduction.....	7
1 Review of ANN, backgammon and development methodologies .....	9
1.1 Artificial neural networks .....	9
1.1.1 The concept of artificial neural network. ....	9
1.1.2 Basics of artificial neural networks.....	10
1.2 Backgammon.....	17
1.2.1 Introduction to backgammon. ....	17
1.2.2 History of backgammon. ....	18
1.2.3 Backgammon variations. ....	18
1.2.4 Game rules. ....	20
1.3 TD-Gammon .....	24
1.3.1 About TD-Gammon.....	24
1.3.2 TD-Gammon's learning methodology. ....	24
1.3.1 Success of TD-Gammon.....	26
1.4 Used development methodologies .....	26
1.4.1 IDEF0.....	26
1.4.2 UML and Object Oriented design. ....	31
2 Design software for managing artificial neural network instances .....	37
2.1 The study of the base implementation of the program .....	37
2.1.1 net.py artifact.....	37
2.1.2 Game.py artifact. ....	38
2.1.3 train.py artifact. ....	40
2.1.4 The work of the source program. ....	42
2.1.5 Profiling the program.....	43
2.2 Training and analysis of ANN.....	43
2.3 Software design .....	45
2.3.1 Structural and functional model of the software product. ....	45

2.3.2	Analysis and detection of requirements. ....	48
2.3.3	Use cases. ....	51
2.4	Design results .....	54
2.4.1	New Game.py artifact.....	55
2.4.2	New net.py artifact. ....	56
2.4.3	New interface.py artifact.....	57
3	Software implementation, testing and obtaining results .....	59
3.1	Selection of development tools .....	59
3.2	Description of the structure and operation of the program .....	59
3.3	Results of the work.....	63
	Conclusion .....	65
	References.....	66
	Appendix A (obligatory). List of illustrative and graphic materials of the GQW.....	69
	Appendix B (recommended). Programmer's manual .....	72
	Appendix C (recommended). User's manual .....	77

## **NOMENCLATURE**

ANN – Artificial Neural Network

RL – Reinforcement Learning

TDL – Temporal Difference Learning

MLP – Multilayer Perception

UML – Unified Modeling Language

ReLu – Rectified Linear Unit

ELU – Exponential Linear Unit

## INTRODUCTION

Artificial neural networks have existed for more than half a century, but the efficiency of their work process and its understanding have not since found a big leap in development. Recently, they have become very popular and show results with which other systems cannot be compared.

The power of artificial neural networks lies in the sheer size of the data on which they are trained. Each year, the amount of these data is growing, and with them the efficiency and accuracy of the statistical model of the neural network.

Artificial neural networks are used in many tasks. One of them is the classification of objects. An example is artificial neural networks for pattern recognition in images. They are trained on a given set of data with a figure and give a statistical estimate of how much the submitted image matches the resulting model along with the hypothetical type of classification. Their seriousness and importance can vary from determining the type of tree to driving a car and diagnosing diseases where human life can be at the cost of error.

In this work, an artificial neural network was used to play backgammon. It is based on the successful example of the Gerald Tesauro – TD-Gammon. He developed several versions of an artificial neural network with reinforcement learning methods that proved success in making difficult decisions. These developments have become the basis for the implementation of a self-learning short backgammon program called TD-Gammon. It achieved mastery of the game, starting with a completely random strategy, constantly playing with herself. Gerald Tesauro made changes to the program and added new components to it. As a result, TD-Gammon reached the level of the game of the world champion. This success made a great contribution to the development of artificial intelligence and inspired many to use such technologies.

The ready implementation of this artificial neural network was trained and analyzed. In the course of research, it was revealed that in the matrix of the network weights, some columns contained only zeros or ones. It was decided to truncate them sequentially and compare the results with the original configuration.

From the foregoing, the goal was set – to determine the criteria for evaluating the results of the operation of an artificial neural network with successive truncation of the weights of «extra» neurons from its matrix.

To achieve this goal, the following tasks were set:

- creation of software for managing instances of the artificial neural network;
- data collection and analysis;
- real-world product testing.



# 1 Review of ANN, backgammon and development methodologies

## 1.1 Artificial neural networks

### 1.1.1 The concept of artificial neural network.

Artificial neural networks are described as computing systems inspired by the biological structure of the brain [1]. They also have neurons, which are computational elements and the connections between them are determined by the so-called weights. The similarity of a real neuron with its model can be seen in Figure 1.1.

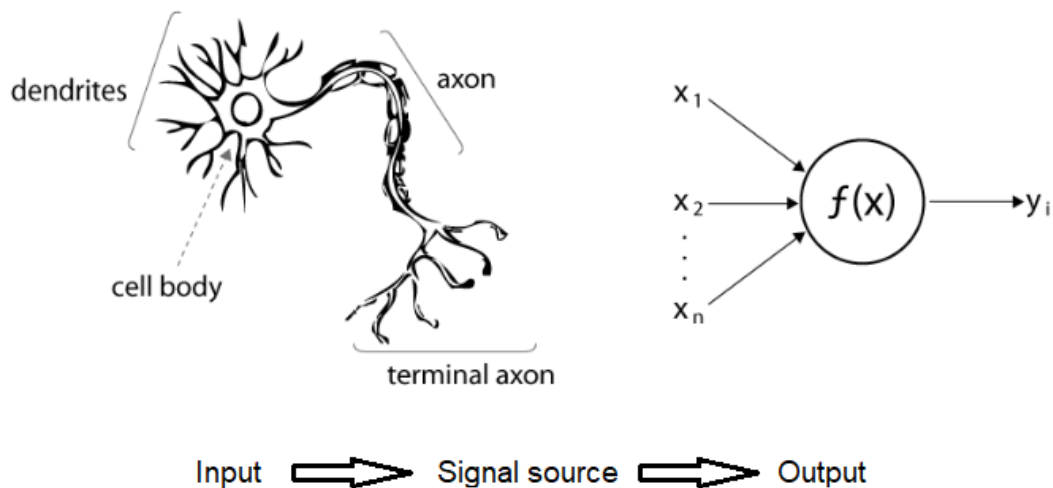


Figure 1.1 – Comparison of a biological and artificial neuron

However, the human brain is very complex and many of its functions are still not understood. At the moment, real and artificial neural networks have the following common functions and characteristics [1]:

- adaptation and training;
- generalization of information;
- massive concurrency;
- robustness;
- storage of data related by associations;

- spatiotemporal data processing.

The initial purpose of artificial neural networks was to solve problems like the human brain. The areas of their application are growing every year and now you can see that they perform tasks that were considered exclusively human activity, especially art [2].

### 1.1.2 Basics of artificial neural networks.

#### 1.1.2.1 Artificial neuron.

An artificial neuron is the main element in the structure of all artificial neural networks. They enter information in the form of weighted data (each input parameter is multiplied by the corresponding weight of the connection). In the body of the neuron, it is summed up, after which the activation function is applied, the bias is added and the result is output. The mathematical description of the above actions is as follows:

$$y = F \left( \sum_{i=0}^m x_i \cdot w_i + b \right), \quad (1.1)$$

where  $m$  – number of inputs of an artificial neuron;

$x_i$  – input value;

$b$  – bias;

$F$  – activation function;

$y_i$  – output value.

A graphically artificial neuron is shown in Figure 1.2 [3].

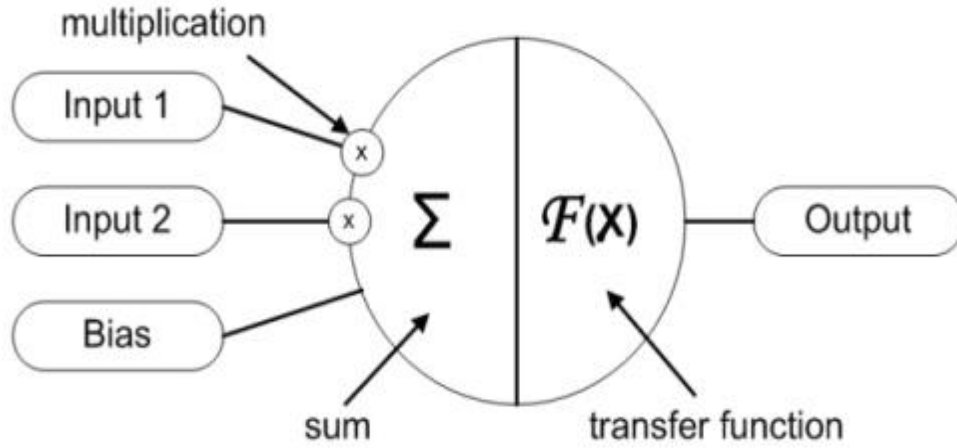


Figure 1.2 – Schematic representation of an artificial neuron

#### 1.1.2.2 Weight matrix.

The concept of weight in artificial neural networks is defined as the coefficient of connectivity between its nodes [4]. Weights represent the strength of these compounds.

The weights of one layer can be considered as matrix elements, where the number of rows will be equal to the number of inputs, and the number of columns to the number of neurons. Then the values of the output vectors for the next layer can be found using matrix multiplication of the weight matrix and the input vector [5]. An example of using the matrix of weights can be seen in Figure 1.3 [6].

$$\begin{aligned}
 z^{(2)} &= \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix} \\
 &= \begin{pmatrix} w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 \\ w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 \\ w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3 \end{pmatrix} + \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix} \\
 &= \begin{pmatrix} w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)} \\ w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 + b_2^{(1)} \\ w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3 + b_3^{(1)} \end{pmatrix}
 \end{aligned}$$

Figure 1.3 – An example of using a weight matrix

When one of the artificial neural network neurons is removed, the weight matrices decrease, losing one column in the layer matrix, where the neuron was located and one row in the matrix, where the output of this neuron was an input for other neurons of the next layer. An illustrative description of this process can be seen in Figure 1.4, where this process is used to increase the «rate of fire» of an artificial neural network [7].

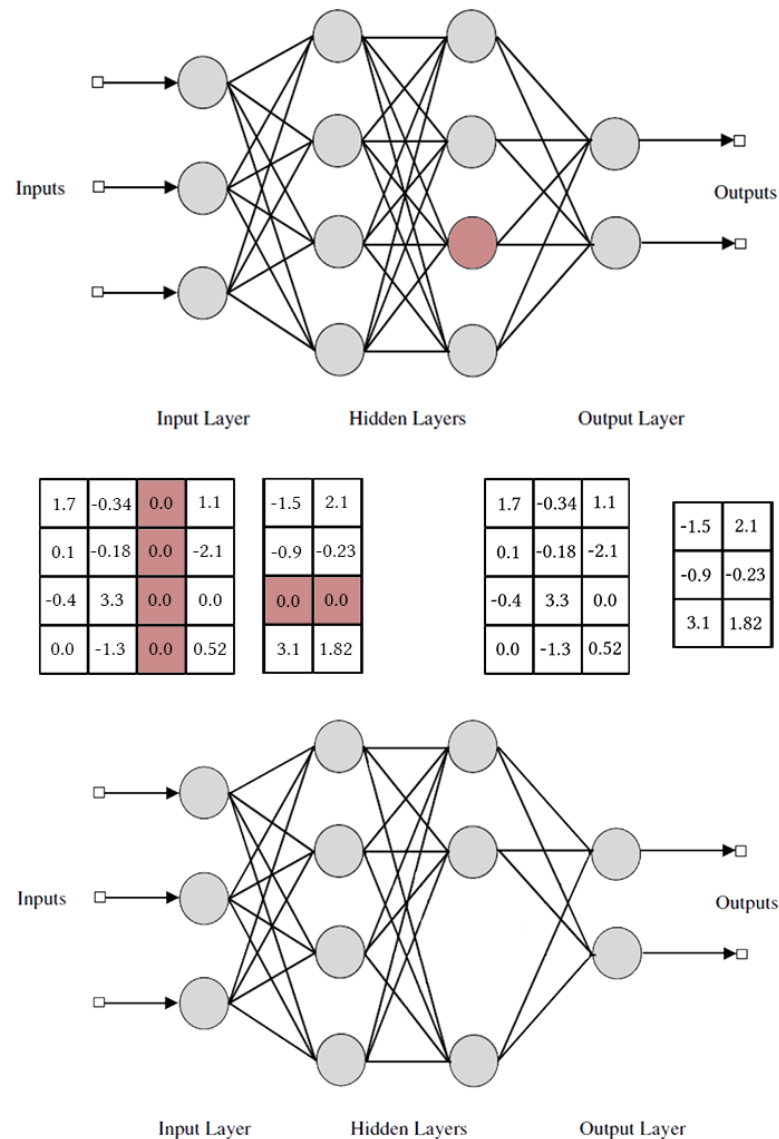


Figure 1.4 – The process of removing a neuron

### 1.1.2.3 Activation functions.

Activation functions or transfer functions are used by artificial neurons in the network to convert input signals to output ones. After receiving all the input data, they are multiplied by the corresponding weights. Then they add up and the activation function brings the amount to a value corresponding to its content. The resulting value goes to the output of the neuron.

The need to use activation functions in artificial neural networks is due to the fact that without them they will be just a simple linear function without the possibility of learning and recognizing patterns in the data. Activation functions introduce dynamism into the system and make it possible to classify complex information from data. To determine system failures or errors, the activation function must be differentiable. This is necessary for training and optimizing the weights of an artificial neural network, for example, using the gradient descent method [8].

The following basic types of activation functions exist:

- sigmoid function;
- hyperbolic tangent;
- Rectified Linear Unit (ReLU);
- Exponential Linear Unit (ELU).

In a research paper [9], these varieties of activation functions were analyzed and their advantages and disadvantages were shown. In the conclusion of the work, it was concluded that there is no better activation function and to decide which one to choose depending on the purpose and structure of the artificial neural network used.

In this work, a sigmoid activation function was used. Its application produces positive numbers between 0 and 1. The graph of values can be seen in Figure 1.5. This function is usually used for data used in network training. Sigmoid activation function is one of the most used [10].

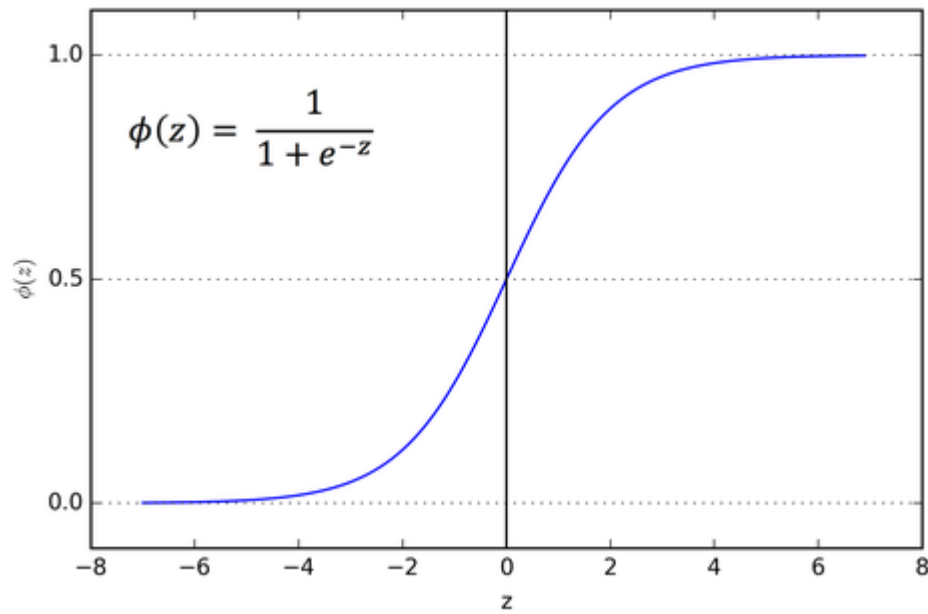


Figure 1.5 – Sigmoid curve and its formula

#### 1.1.2.4 The process of training an artificial neural network.

To train an artificial neural network, various processes and methods are used. Under the training process, we can understand the regulation of the values of weights to obtain the desired results. Learning methods can be divided into two categories: supervised and unsupervised. Their main difference is the availability of a training sample. It is used in artificial neural networks with training with a teacher. Before starting work, the network learns from those provided until its output meets the set requirements. In contrast, learning without a teacher is not aimed at a specific result, and the network itself tries to recognize implicit patterns in the input data, constantly adapting to them [11].

An example is the use of cost functions [12]. Having the initial and required data, the artificial neural network generates predictions and finds the difference between them and the given requirements. By minimizing this function, the predictions become more accurate and closer to the answer. Figure 1.6 shows a graph of the gradient descent algorithm, on which you can see an increase in the accuracy of predictions [13].

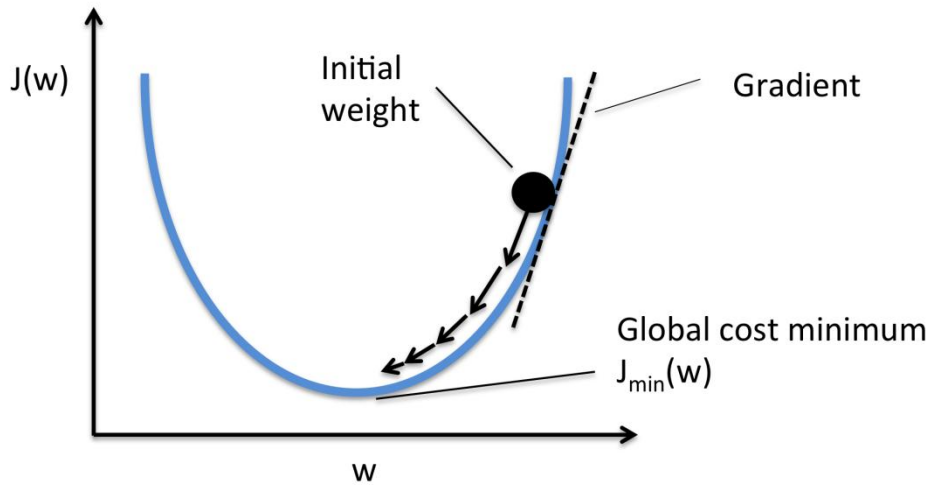


Figure 1.6 – A graph of the cost function

#### 1.1.2.5 Classification of artificial neural networks.

The classification of artificial neural networks is shown in Figure 1.7.

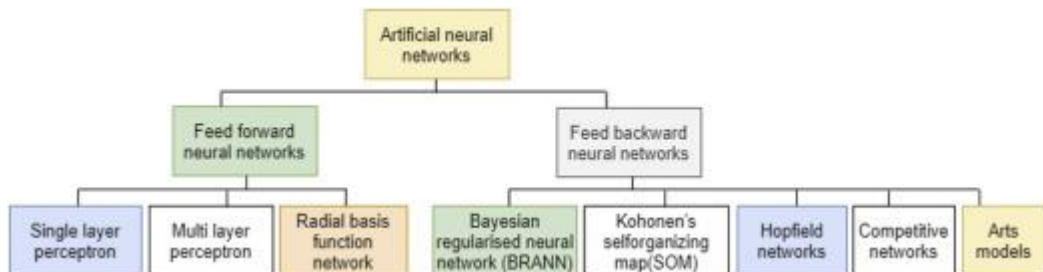


Figure 1.7 – Classification of artificial neural networks

In [14], artificial neural networks are divided into feedforward and feed-backward neural networks. Feedforward artificial neural networks are more similar to the structure of the human brain. Information comes and goes through them in only one direction: from input to output neurons. An example of this architecture is shown in Figure 1.8.

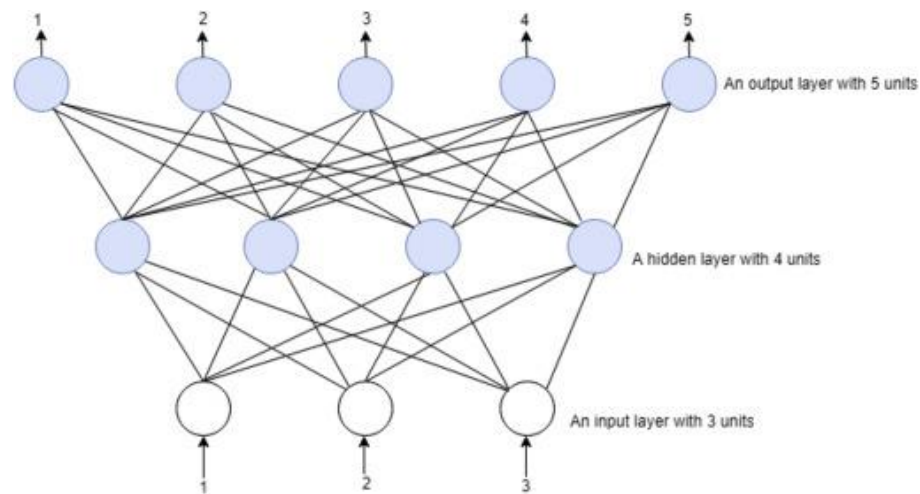


Figure 1.8 – Feedforward network with two layers of neurons

Feed-backward artificial neural networks contain internal states that are used to process input sequences. An example of the structure is shown in Figure 1.9.

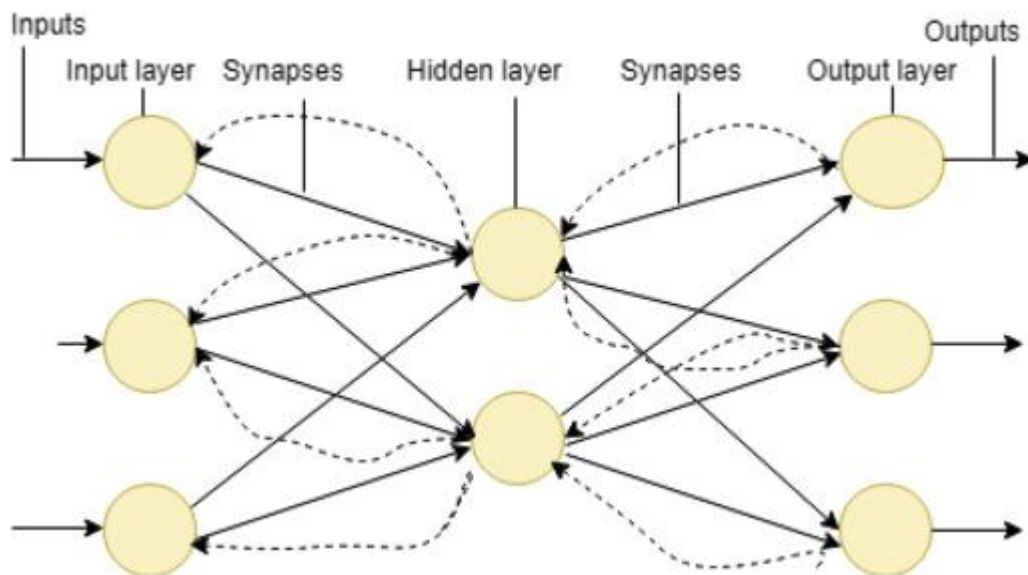


Figure 1.9 – Feed-backward neural network

#### 1.1.2.6 Concepts black box, white box and gray box.

These concepts in [15] are described as follows. The black box describes the models where nothing is known except its inputs and outputs. Artificial neural networks are considered black boxes. In contrast to the black box, the white box is completely open and



known, and its solutions are understood and can be explained. The gray box is a combination of black and white. In it, relationships and dependencies are determined through data analysis and some of their functions are not fully understood. The visualization of these models can be seen in Figure 1.10.

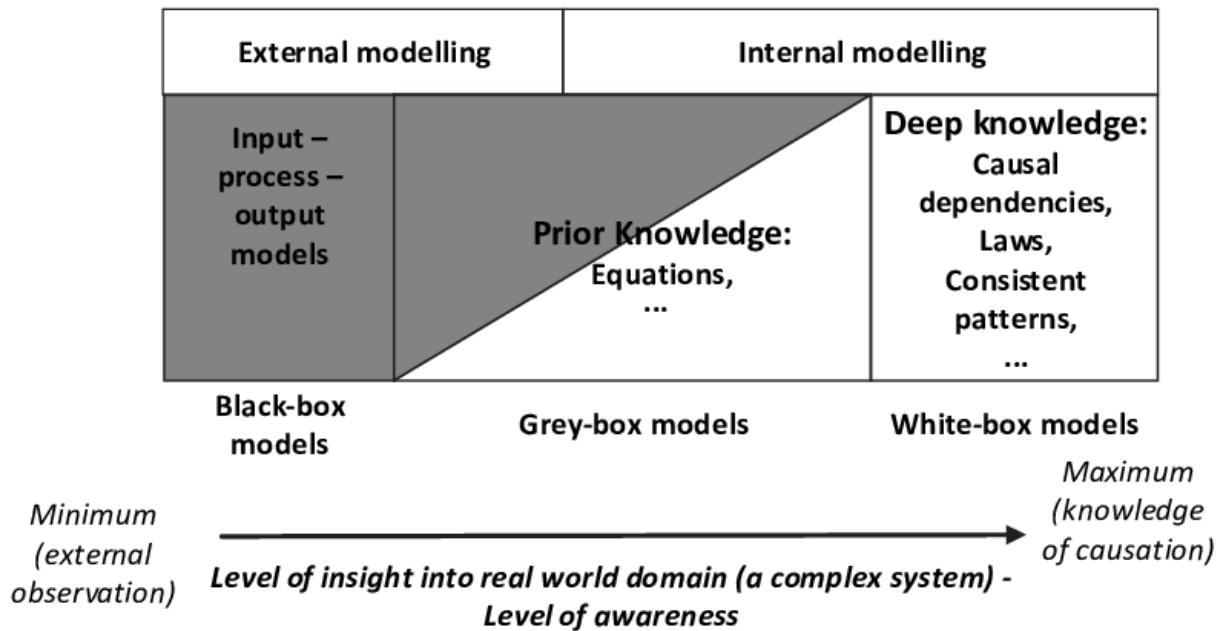


Figure 1.10 – Visualization of concepts black, white, gray boxes

## 1.2 Backgammon

### 1.2.1 Introduction to backgammon.

Backgammon is a board game in which two players participate. They are in a «race» between themselves and everyone is trying to get out of the game first. The one who first takes out all the pieces from the board is the winner. In backgammon, unlike other strategic board games, there is a luck factor that is of particular interest to computer science.

To play backgammon, you need to have a board, two sets of 15 drafts, two dice, a cup for a rolls and a doubling cube. This complete set can be seen in Figure 1.11.



Figure 1.11 – Board for playing backgammon

### 1.2.2 History of backgammon.

Backgammon is considered one of the oldest games and their origin dates back to about 3,500 years ago. Central Asia is the supposed homeland of backgammon. The first mention of a game similar to the rules of backgammon comes from Byzantine Greece, which describes the game of Emperor Zeno. The difference from the modern version was the presence of an additional cube and another initial arrangement of checkers on the board [16].

It is believed that the name of the game comes from the Welsh words bac (bach) and gammon (cammaun), which can be translated as “little battle”. There is also a version about the English origin of the term, where it had the meaning of “back game”. In confirmation of this, historians speak of chessboards where the backgammon board was on the back side.

### 1.2.3 Backgammon variations.

Backgammon has many variations. Many of them belonged to individual geographic regions and brought diversity to the game. Usually the changes affect the initial positions of the pieces, possible moves and add values to the dice, but in some cases even change the direction of movement of the pieces on the board.

The following three variations are given as an example:

- nackgammon: in the second paragraphs moved one checker from the slots, where there are 5 each [17];

- longgammon: all checkers are transferred to the first slots [18];
- hyper-backgammon: each player has only three checkers, one in the first three slots [19].

Examples of boards for each of the variations can be seen in Figures 1.12 – 1.14.



Figure 1.12 – Nackgammon

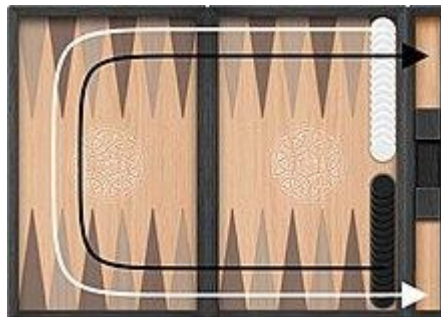


Figure 1.13 – Longgammon

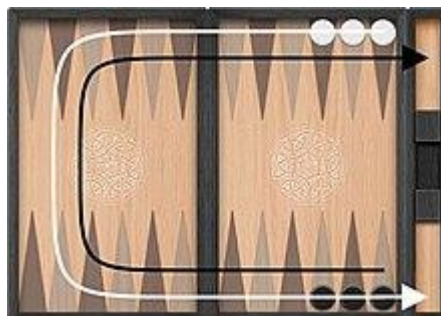


Figure 1.14 – Hyper-backgammon

#### 1.2.4 Game rules.

The rules of the game were analyzed and written from the source [20].

##### 1.2.4.1 Initial placement.

Backgammon is a game for two players. It passes on a board consisting of twenty-four narrow triangles, which are called points or slots. They alternate in colors and form 4 groups of 6 triangles. These groups have the following names: home, home of the enemy, outer board, outer board of the enemy. Homes and outer boards are separated by a bar – a bar that divides the playing field. An example arrangement is shown in Figure 1.15.

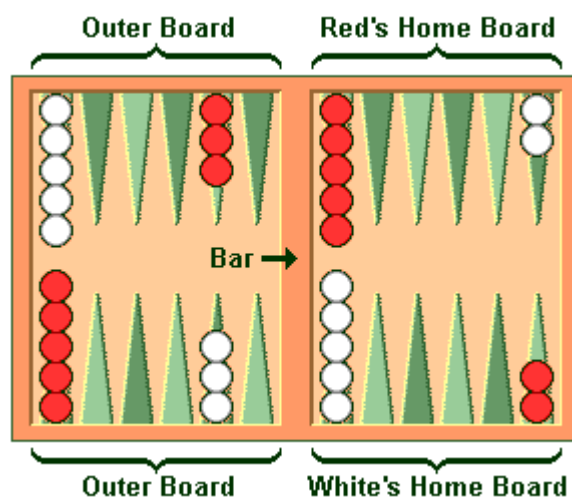


Figure 1.15 – Board with checkers in the initial positions

The points are numbered from 1 to 24. For each player, the numbering starts from the outermost point of their home. Each player has 15 checkers, usually black or white. They are arranged in the following order: two drafts on the twenty-fourth point, five on the thirteenth, three on the eighth and five on the sixth for each player.

Each also has a pair of dice and a cup for shaking them. A doubling cube has the numbers 2, 4, 8, 16, 32 and 64 on the sides. It is used to track the multiplicity of bets during the game. It can be seen in Figure 1.16.



Figure 1.16 – Doubling cube

#### 1.2.4.2 Checkers movement.

Before the start of the game, each player rolls one die. This determines the player who will make the move first. If the same numbers fall out, then the die rolls again. The first player makes a move in accordance with the two dice thrown. The next move is made by the second player and rolls two dice at once.

The resulting combination of values of the dice determines how many points the player can move the checkers. Checkers always move towards the point with the lowest number. The movement of the pieces is shown in Figure 1.17. The movement is determined by the following rules:

- a checker can only be moved to an open point where there is no more than one enemy checker;
- each die roll defines a separate movement. For example, if a combination of 5 and 3 occurs, then a player can move one piece 3 points forward, and the other 5, if these points are open. It is also possible to move only one checker by the combined number of moves ( $3 + 5 = 8$ ), but only if the intermediate point (after moving by 3) is also open;
- if a player has a combination with two identical numbers, he can use the dropped number 4 times in any combination;
- the player must use all the combinations obtained, if possible. If it is possible to use only one number from two, then the player must choose the larger one. If no move is possible, the player skips his turn.

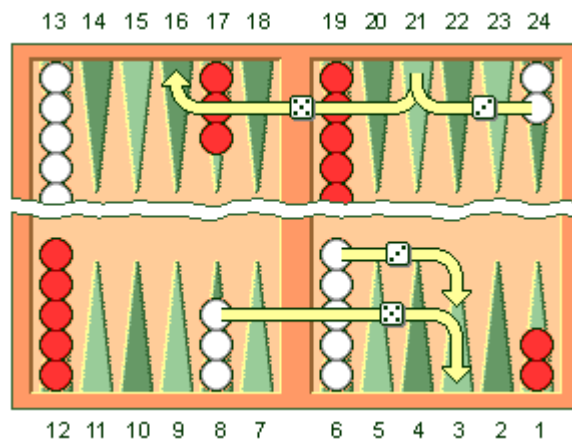


Figure 1.17 – Two variants of movements

#### 1.2.4.3 Hitting and entering.

The point where only one checker is located is called a blot. If a checker of a different color switches to the blot, then the checker standing on the blot will be hit and placed on the bar.

If there is one or more checkers on the bar, the player must bring them to the opponent's home. Checkers are deduced from the bar by throwing dice. The player can bring the checker to the point corresponding to one of the numbers of the dice, provided that it is open. For example, in Figure 1.18, it is shown that the player throws a combination of 4 and 6 and he can move the checker to either the fourth or sixth point of the opponent.

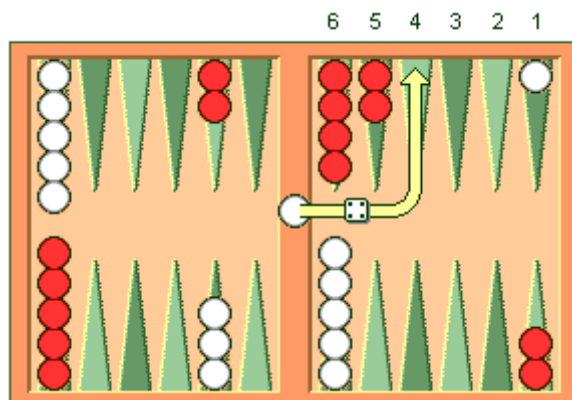


Figure 1.18 – The white player throws out 6–4

If none of the possible points is open, then the player skips his turn. When all checkers are removed from the bar, if the player has unused numbers, then he is obliged to use them on any checkers.

#### 1.2.4.4 Bearing off.

Bearing off checkers from the game means removing them from the board in accordance with the thrown combination after all the checkers of the player are in his home.

After all the checkers of the player are moved to his home, he can start throwing them away. Throwing is possible when one of the numbers of the dice rolls corresponds to the number of the item on which the checker stands. Figure 1.19 shows bearing off when rolling 6.

If the points whose numbers correspond to the numbers thrown out do not have checkers, then the player must make a move with the checker, which is located at the point with the highest number. If there is no such checker, then the player can and must bear off the checker off the point with the highest number. A player is not required to bear off checkers if he can make any of the moves that complies with the rules.

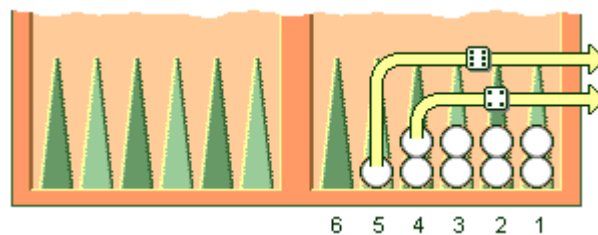


Figure 1.19 – White player throws checkers

If the checker was knocked to the bar during the throwing, the player must bring it to his home before continuing the throwing process. The first player who succeeds in bearing off all 15 of his checkers wins the game.

## 1.3 TD-Gammon

### 1.3.1 About TD-Gammon.

TD-Gammon is an artificial neural network for playing backgammon. It is able to learn while playing with herself. This ANN was developed in 1992 by Gerald Tesauro. The process of learning begins with random weights, but with an increase in the number of games played, the network achieves a strong game level [21]. The latest versions of TD-Gammon are able to compete with champions, and in 1979 the program defeated the world champion in backgammon by a margin of 7 points. This was the first breakthrough when a human-created program defeated a champion [22].

### 1.3.2 TD-Gammon's learning methodology.

The TD-Gammon program is based on an artificial neural network with MLP architecture. Its structure is shown in Figure 1.19. This structure is widely used in reinforcement and teacher learning algorithms, which can be perceived as a normal nonlinear approximation function. The result of this program is calculated by feedforward flow, from input to output neurons. The data process also passes through a hidden layer of neurons. Data moves according to connections between neurons and is converted by connection coefficients – weights. This neural network uses a sigmoid activation function, which brings the values of the inputs to one interval.

For the MLP architecture, the learning process consists in changing the values of weights using formulas until the results of the operation of the artificial neural network meet the required ones. This approximation of weights can take a tremendous amount of time and calculations if the goal is to determine the global optimal set of weights. However, local kits can be satisfactory and be successfully applied to problems in the real world. There are many algorithms and training techniques for finding them, such as reinforcement learning, for example.



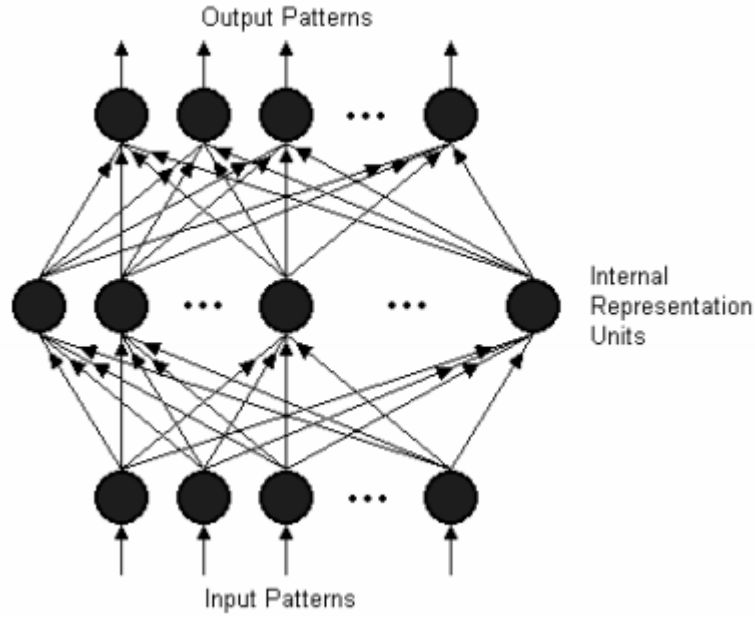


Figure 1.20 – Multilayer perceptron used in TD-Gammon

The process of training an artificial neural network TD-Gammon is described as follows: a sequence of positions of checkers on the board and the state of the game are input to the network. Positions enter the network as a vector with encoded values. A new vector is formed for each move of any player and the network gives a prediction, deriving as a result a vector of two values: the chance of winning white and the chance of winning black. The TDL algorithm is applied to each sequence, as a result of which the network weights are adjusted. Formula of TDL is represented as follows:

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^t \lambda^{t-k} \nabla_w Y_k, \quad (1.2)$$

where alpha is a constant, which is usually defined as the speed of learning. W is the weight vector of a given neural network, and Y is the output vector.

Lambda is a parameter responsible for determining errors. With lambda = 0, feedback does not occur, and with lambda = 1, the error is processed by the network and fed back. The lambda parameter provides an offset between these two extreme cases.

At the end of each game, a  $z$  reward is generated based on the outcome of the game. The difference between  $z$  and  $Y$  in the equation is responsible for changes in the values of the weights.

During training, the neural network itself chooses the moves for each of the players. For each move, the program determines the possible actions and selects one of them that meets the highest expectations. This process occurs from the very beginning of the game and even at the start of training, when the initial values of the weights are absolutely random. In this regard, in the initial stages of training, the number of moves can exceed a thousand, while the usual party, which is played by people, takes 50–60 moves [23].

#### 1.3.1 Success of TD-Gammon.

The study [24] says that success in training TD-Gammon is not explained by training methods with reinforcements or the TDL algorithm, but by the dynamism of the game itself and gradual improvement when playing with itself. This evolutionary approach has made TD-Gammon a very important step in the development of this area. The game model turned out to be simple, where the environment allowed the network to learn and improve in an endless cycle without changing the rules. The network itself responds to its actions and makes improvements.

### 1.4 Used development methodologies

This section describes the methodologies, approaches, and tools used to develop the software.

#### 1.4.1 IDEF0.

To apply this methodology, its standard issued by the US Department of Defense was studied [25].

The abbreviation IDEF0 stands for the definition of integrations for modeling functions. This methodology was developed based on SADT, created by Douglas T. Ross. IDEF0 includes:

- graphic modeling language;
- description of methodology for model development.

IDEF0 is used to determine the requirements and functions for a new system being developed, as well as to analyze existing ones. Models that are created using IDEF0 consist of diagrams in which the main components are functions and data. In the diagrams, the functions are shown as boxes, and the data as arrows.

Next, consider using the main elements.

#### 1.4.1.1 Boxes.

In diagrams, boxes perform the function of describing a process. Each box contains text – the name of the function or process that it defines and the number. The numbers in the boxes are necessary to be able to refer to them in the description. The standard view of the box is shown in Figure 1.20.

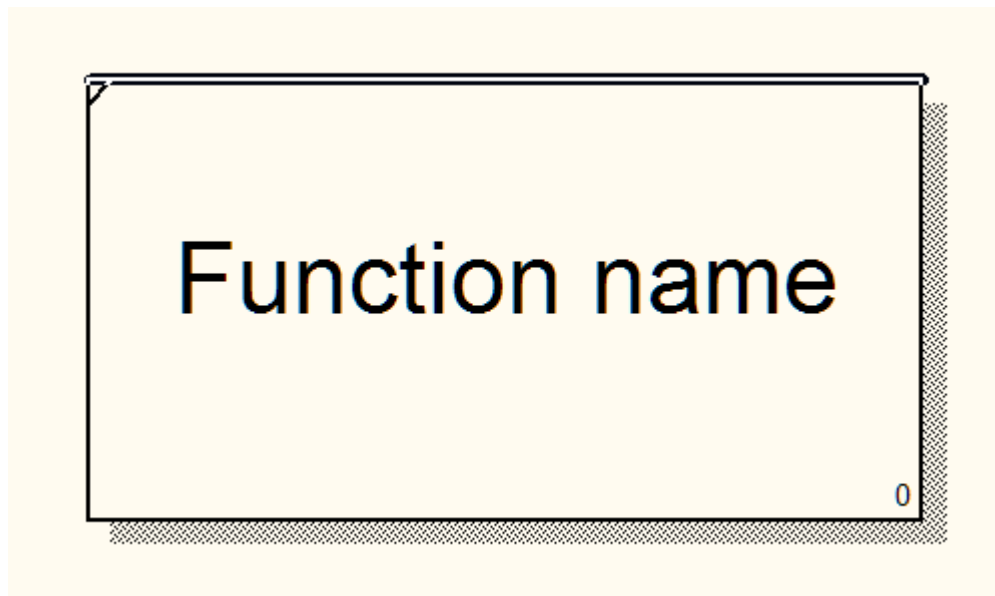


Figure 1.21 – Box in diagram

For boxes, the following design rules are distinguished:

- the boxes must be large enough to contain the function name;
- boxes must be large enough to accommodate boxes must have a rectangular shape;
- the sides of the boxes must be solid lines.

#### 1.4.1.2 Arrows.

The arrows define the data in the model and how they are used by the functions. In this case, the arrows in the diagrams do not display the sequence of the flow, but only the relationship between the objects. Arrows can be of different shapes. The Figure 1.21 shows four types of arrows:

- straight;
- with right angle corners;
- with branches;
- merging.

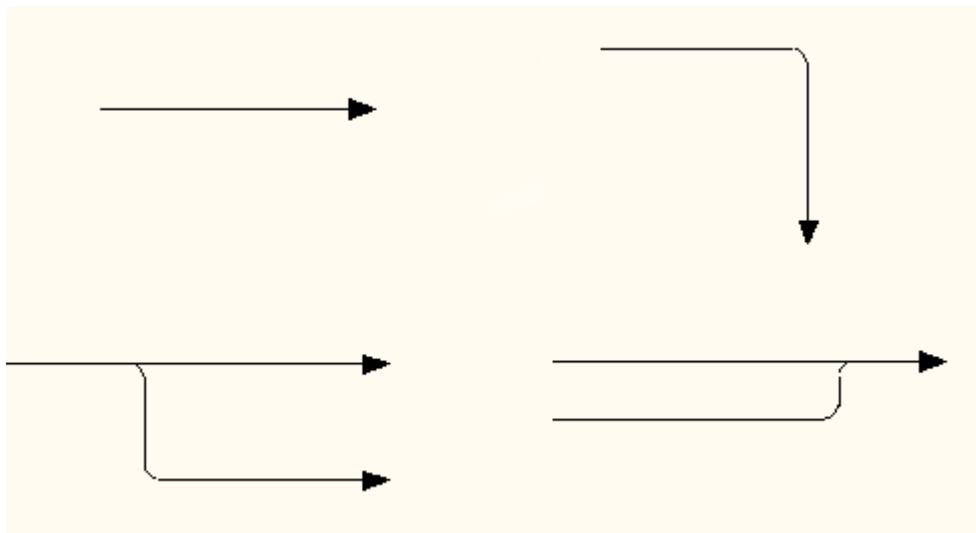


Figure 1.22 – Types of arrows in the model

Arrows must be designed according to the following rules:

- turning arrows should only be bent at right angles;
- arrow lines must be solid;
- arrows can only be horizontal or vertical;
- arrows should touch the sides of the box but not enter it;
- arrows cannot be connected to the corner of the box.

### 1.4.1.3 Relations between arrows and boxes.

The value of the connected arrow changes depending on the side of the box. The four sides of the box reflect the four different roles of the arrows. The function inputs are indicated on the left side. Data that points to this side will be consumed by the function and modified.

The arrows pointing to the top of the box have a control role. Control indicates how the work in the function will be performed and under what conditions. This role is necessary to obtain the correct results of the function. The results of the function are displayed on the left side of the box in the form of outgoing arrows. The output of the function is the input data over which the function processed.

The last bottom side of the box describes the arrows with the role of mechanisms. Mechanisms contribute to the function itself.

The standard arrow positions in the diagram are shown in Figure 1.22.

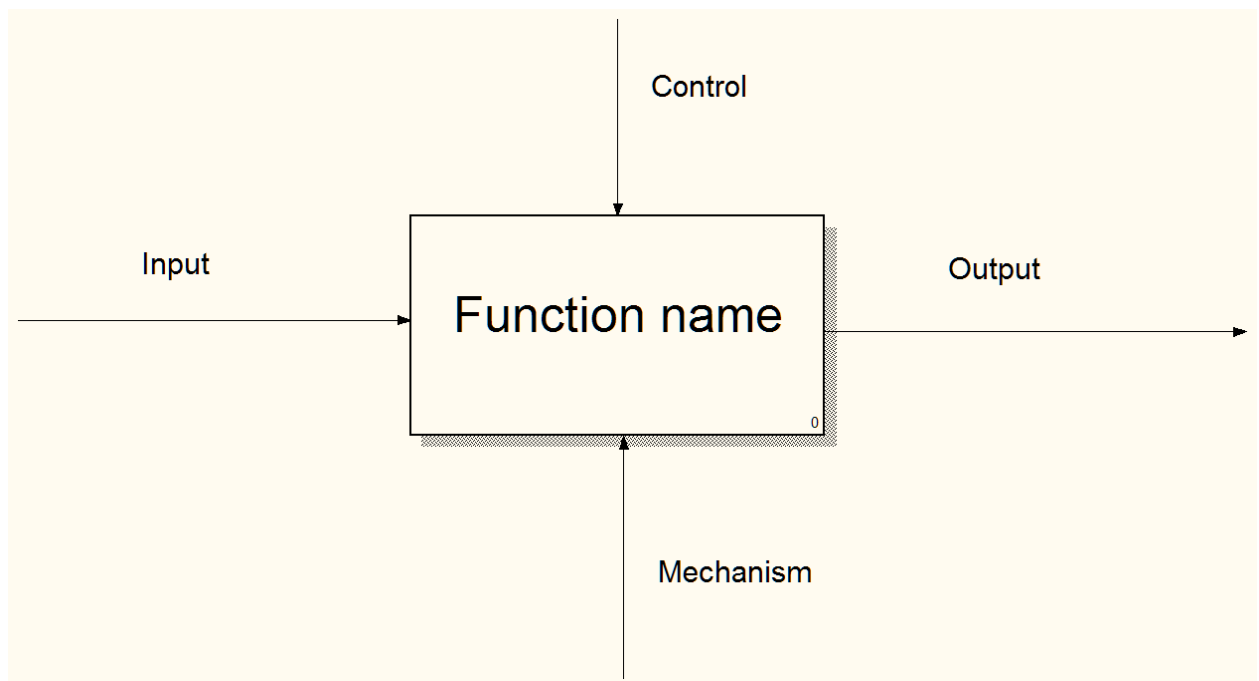


Figure 1.23 – Standard positions of arrows in the diagram and their roles

#### 1.4.1.4 Top level diagram.

The top-level diagram is present in all models and reflects the object of study as a whole. This diagram is indicated by A-0 during construction. It contains only one box and arrows indicating the external interfaces of interaction with the system.

A-0 is a context diagram that reflects the point of view and purpose of the model. Her example can be seen in Figure 1.23. The point of view is the perspective from which they look at the model and shape its context. The goal indicates the reasons for the creation of the model determines the structure of the model. Point of view and goal are necessary to create constraints and clarify the design process.

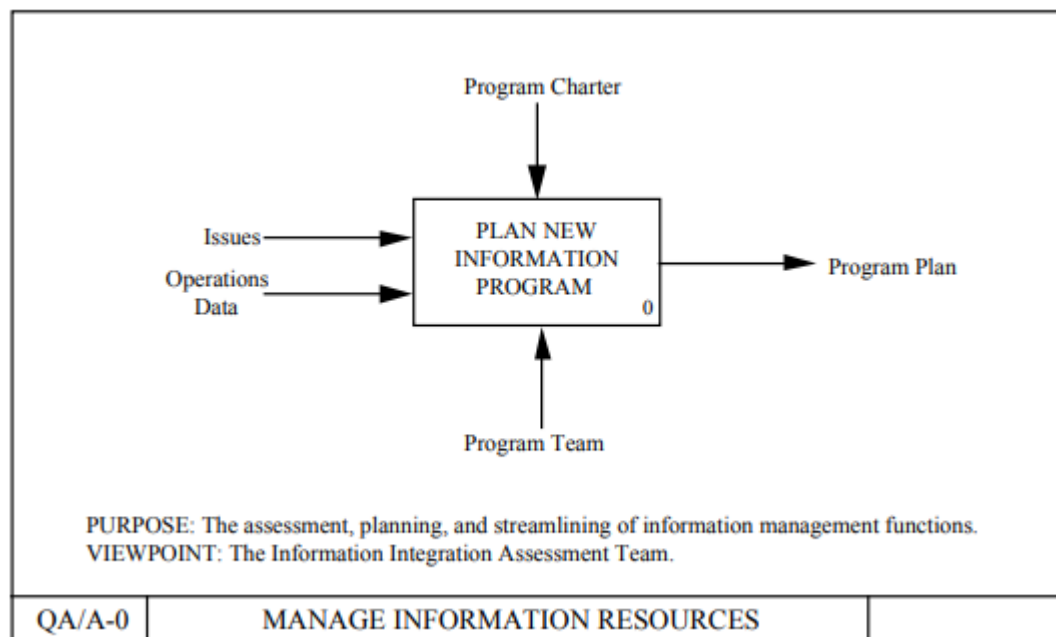


Figure 1.24 – An example of a top-level diagram

A context diagram can be decomposed into subfunctions to produce a new diagram. It will reflect the internal processes of the model. Each of these subfunctions can also be decomposed. The decomposition process can be continued until a complete understanding of the model is reached, and the point of view is not fully disclosed. This hierarchical view is a top-down approach and controls the level of detail of the model. Program decomposition levels are shown in Figure 1.24.

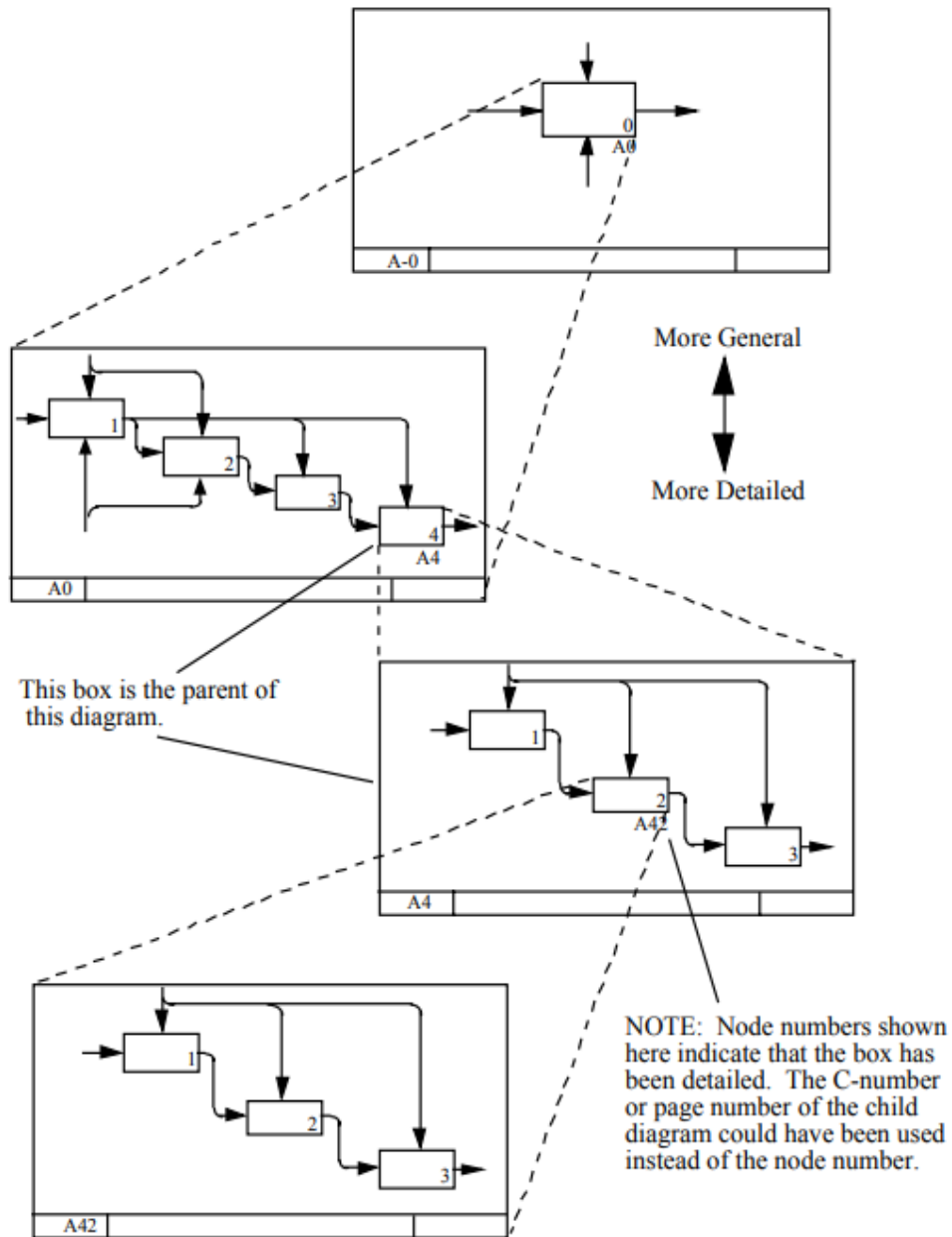


Figure 1.25 – Structure of decomposition

#### 1.4.2 UML and Object Oriented design.

In connection with the development tools used for software design, sources [26] and [27] were also studied, which describe the UML modeling language and object-oriented analysis and design.

##### 1.4.2.1 Software development process.

The software development process can be described as the transformation of customer requirements into the final system. Requirements can be both functional and

non-functional and should describe what should be done, and not how. Requirements are often incomplete and vague to understand. Therefore, the collection and analysis of requirements is a mandatory process when starting the development of software products.

The collection and analysis of requirements leads to the creation of requirements specifications. This document reflects the basis for building a system. To create it, you need the following processes:

- domain study;
- definition of requirements;
- classification of requirements;
- validation;
- feasibility study.

Their interaction and relationship among themselves is shown in Figure 1.25.

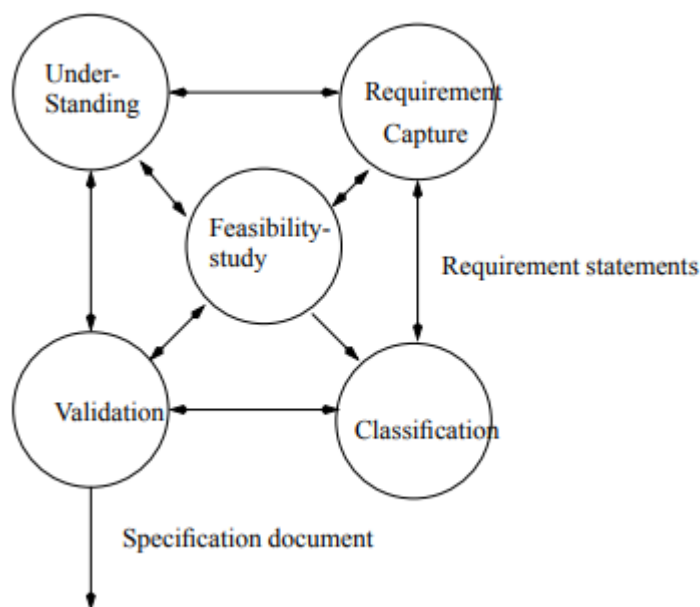


Figure 1.26 – The requirements analysis process

The system functions of the program clearly display all the system requirements for themselves. When forming requirements, each function should continue the sentence “The system should ...” to exclude non-functional components.



To prioritize, functions should be divided into groups. It also contributes to their accounting, so you can make sure that none of them were missed. System functions are divided into two groups:

- evident functions;
- hidden functions.

Evident functions are known to the end user and he is able to use them. Hidden functions are responsible for the technical part and their implementation is imperceptible to the user. Most often, hidden functions are overlooked.

#### 1.4.2.2 Modeling with UML.

The unified modeling language UML is intended for visualization and design of various systems and processes. Its first standard was published in 1997 by the Project Management Group. It is particularly well suited for systems that use an object-oriented programming style.

Usually, UML is divided into three types. In the first case, UML is used to create sketches. These are small diagrams that describe only some parts of the system. This approach is used in forward engineering and reverse engineering processes. A code is generated from the created diagram, or a new diagram is drawn from the existing code.

It is also common to use this modeling language as a complete system plan. Unlike sketches, the plan of the system is very detailed and complete. Often they are engaged in a system designer, but usually this role is played by a senior programmer. He designs the system, after which the team of programmers begins work.

After much experience with UML, people begin to perceive it more as a programming language. To do this, tools were invented that generate program code from the constructed diagrams.

The UML 2 standard describes 13 different diagrams, which are divided into two groups (Figure 1.26.). In many cases, chart data elements can be used freely in each other.

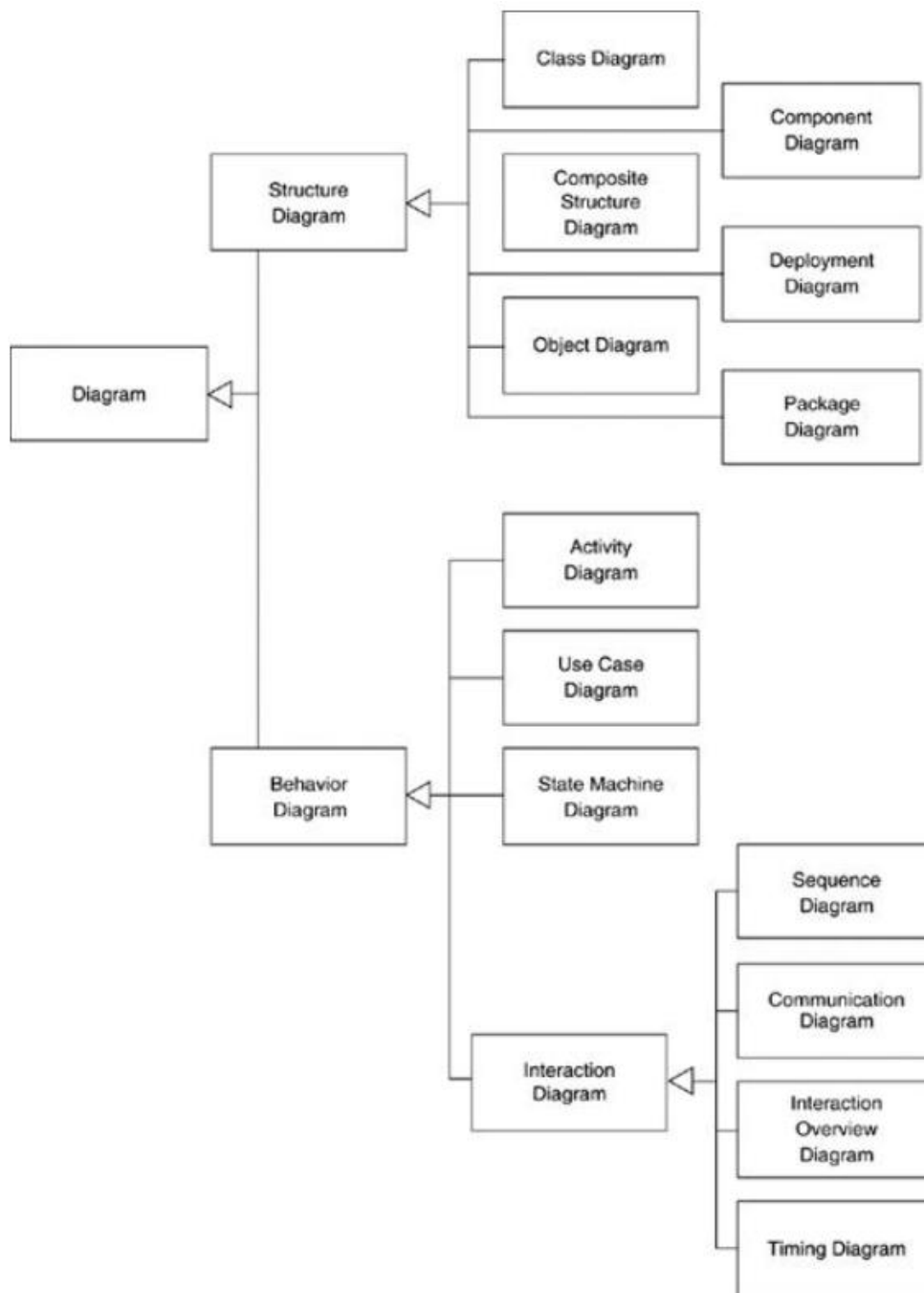


Figure 1.27 – Classification of UML diagrams

The main elements of UML are:

- things;
- relationship;
- diagrams.

Things, in turn, are divided into the following categories:

- structural;

- behavioral;
- grouping;
- annotations.

Structural things include classes, interfaces, collaboration, use cases, components, and nodes. Classes describe object templates and their properties. Interfaces are used to describe sets of operations. Collaborations show interactions between objects. Use cases are actions that the system performs. Components indicate the physical parts of the system. Nodes show the physical elements that exist when a program runs. These elements are shown in Figure 1.27.

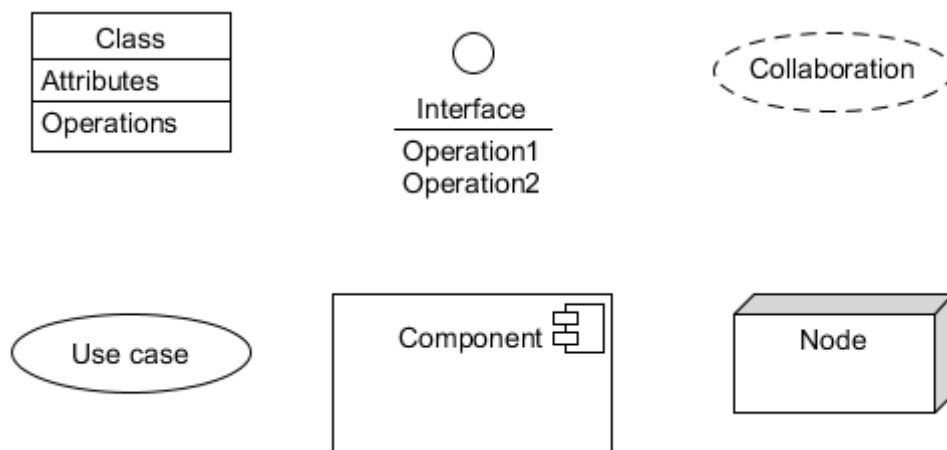


Figure 1.28 – Structural things in UML

Things of behavior are responsible for the dynamic operation of the system. Two things of this type are distinguished: interaction and state machine. They are depicted in Figure 1.28. By interaction we mean the exchange of information between elements. States show the state of the object.

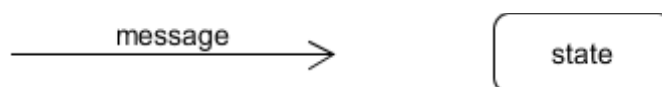


Figure 1.29 – Behavioral things in UML

There is only one grouping thing – it's a package. It connects chart elements together. There is also only one thing to annotation – a note. Notes are needed for commenting and describing model elements. Two of these things are depicted in Figure 1.29.

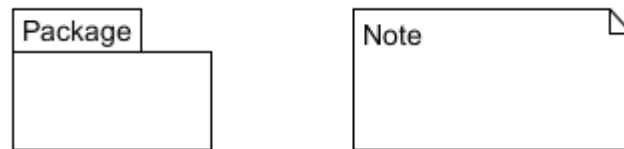


Figure 1.30 – Grouping and annotation things in UML

Relations are shown as a set of arrows (Figure 1.30.). They can be considered the most important elements of UML. They show the association and connection between them and demonstrate the functionality of the system. There are four types of relationships. Relationships of dependence, association, generalization and implementation. A dependency describes the relationship between objects in which a change in one object leads to a change in another. The association shows the relationship between the elements. Generalization relates to particular and general cases. The implementation points to the object in which the function used exists.

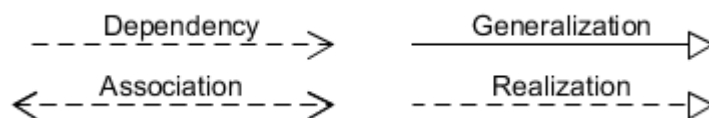


Figure 1.31 – Relations in UML

## 2 Design software for managing artificial neural network instances

### 2.1 The study of the base implementation of the program

The ready implementation of the artificial neural network was taken from an open source [27]. It was created in 2017 and consisted of three script files in the Python programming language. The file structure is shown in shown in Figure 2.1.

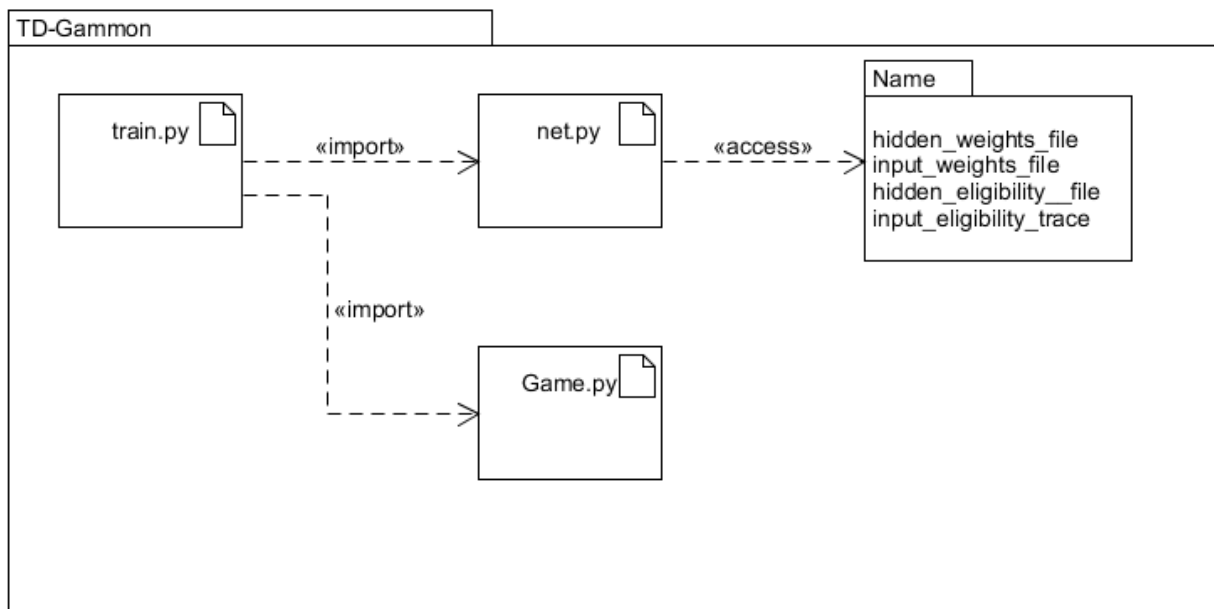


Figure 2.1 – UML diagram of initial implementation packages

#### 2.1.1 net.py artifact.

The net.py file contains the Net class, which implements an artificial neural network with one hidden layer of 40 neurons. It has 198 outputs in which the state of the game at the time is encoded. Two output neurons give the result in the form of the probability of winning a white and black player. These probabilities are not opposite and can be equal. To visualize the class, a model was created in the UML notation, which is shown in Figure 2.2.

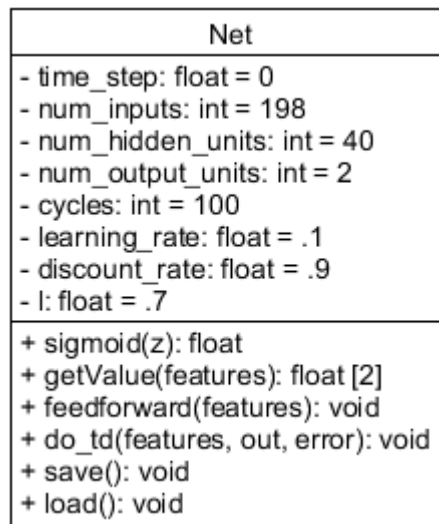


Figure 2.2 – UML diagram of the Net class

The `learning_rate` variable is responsible for the alpha parameter in the learning process. The gamma parameter is the variable `discount_rate`, and the lambda is 1. All parameters were determined as used by Tesauro himself. The sigmoid activation function is described in the sigmoid function. Feedforward changes the weights of the neural network and `do_td` starts the learning process using the TD algorithm. The save function unloads the weights into files in the data folder and load reads them from there.

#### 2.1.2 Game.py artifact.

The `Game.py` file contains an implementation of the Game class. He simulates the rules, possible actions and conditions of the game, but the game process is not described there. The class properties and methods are shown in Figure 2.3 in the UML class diagram.

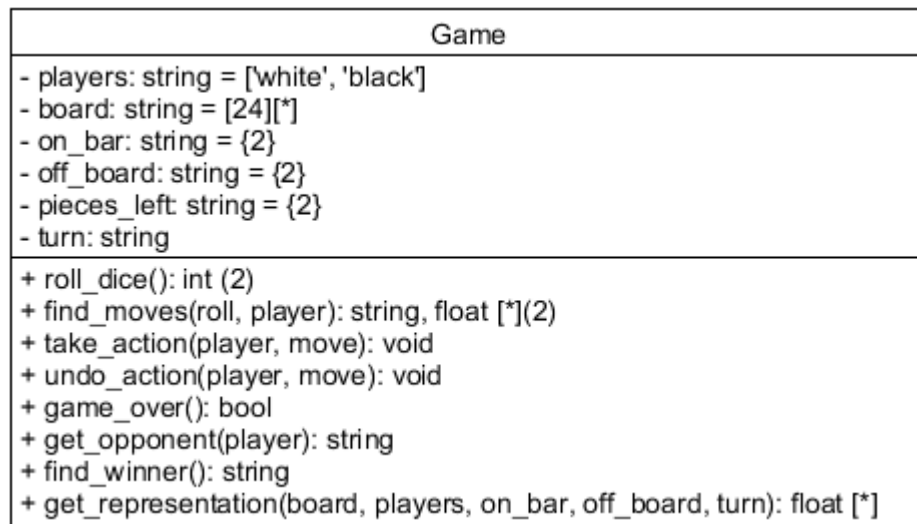


Figure 2.3 – UML Game class diagram

Checker positions on the board are stored in the variable board. When creating a class object, they are arranged according to the rules of the game. The variables on\_bar, off\_board and pieces\_left are dictionaries with key names in the form of player names as in the players variable. They show how many drafts the players have on the bar and offside and how many drafts each player has.

The roll\_dice method is responsible for throwing dice and returns 2 values from 1 to 5. The find\_moves method is used to determine possible moves when dice are thrown. It returns all available moves in the form of a list with elements consisting of two values. They can be numbers indicating the number of values, or actions, for example, as a checker output from a bar.

The take\_action and undo\_action methods are responsible for committing player actions. They pass actions obtained from the find\_moves method. In take\_action, checkers move around the board, and undo\_action cancels any of the actions and returns the state of the board one move before it.

You can find out if the game is over using the game\_over method. If it is completed, then you can find the winner using the find\_winner method. He returns the winner, if any.

Inputs for an artificial neural network are generated using the get\_representation method. It encodes the state of the game as follows:

- the first 192 entries consist of groups of 4 for each item and for each player. The first three moves correspond to the number of drafts from 0 to 3. So, if there are no drafts on the point, then the combination will be 000, if there is one checker 100, with two 110 and three 111. The fourth input with quantities less than 4 is 0. With large quantities the fourth input is calculated from the equation  $(n - 3) / 2$ , where  $n$  is the number of drafts;
- the following two inputs determine the number of checkers on the bar for each player. Data is written as  $n / 2$ , where  $n$  is the number of checkers;
- two more inputs are needed to indicate the checkers that have left the game. Recording takes place in the form  $n / 15$ , where  $n$  is the number of checkers;
- in the last two, the player who made the move is encoded. For white 01 and for black 10.

### 2.1.3 train.py artifact.

The training cycle and the game process are contained in the train.py file. Possible options for working with the program must be entered into the file manually. So, for example, to change the number of learning games, you need to change the number in the loop condition. To understand the processes of the program, an activity diagram was built; it can be seen in Figure 2.4.



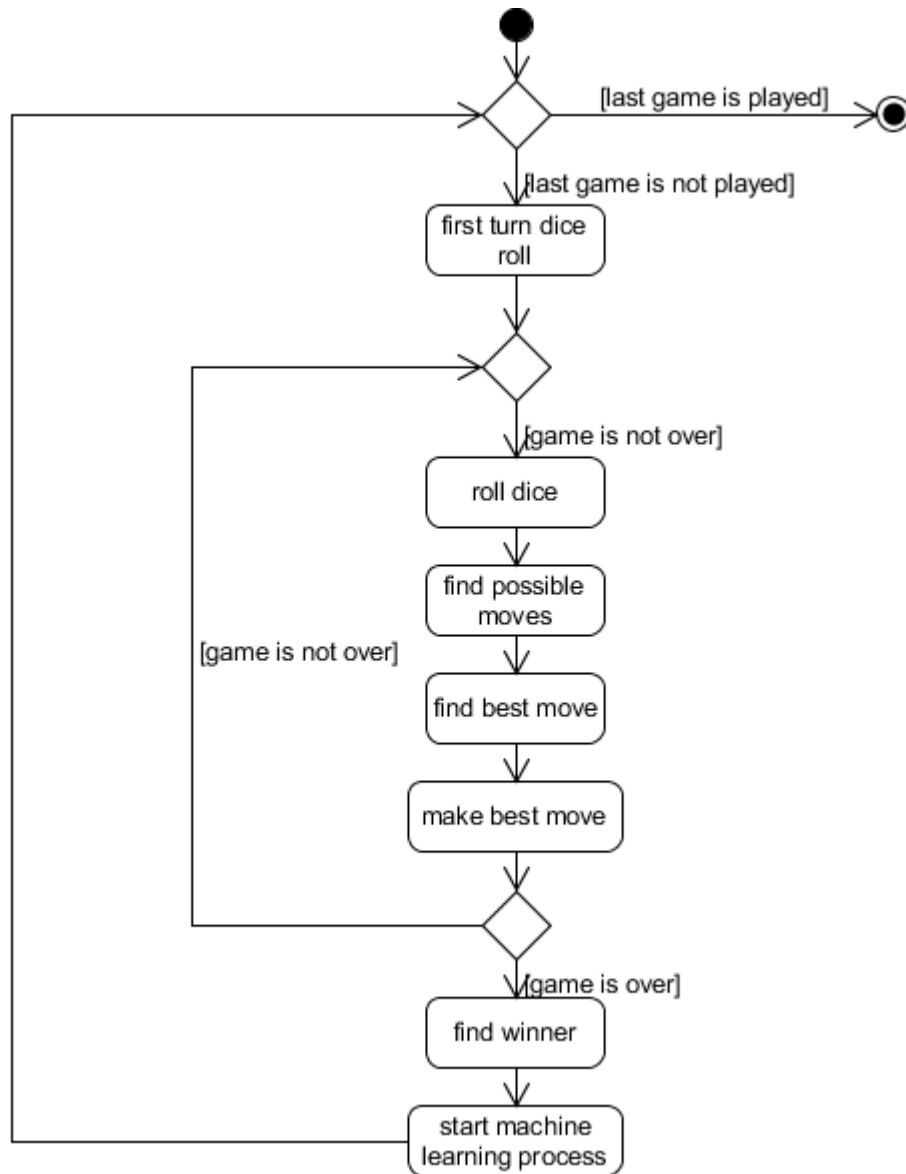


Figure 2.4 – UML activity diagram of the train.py file

For the program to work, the classes Net and Game are imported into the train.py file. To start the game, dice are thrown and the player with the first move is determined. Using the combination obtained, the possible moves are determined.

The best move is determined by the following algorithm: in the loop, an action is taken on each found move and a representation of the state of the game by the neural network is recorded. After which the action is canceled and another is tested. At the end of processing all actions, a move with the best probability of winning is selected.

The process will continue until the game is over. Next, the winner is found and displayed, and the machine learning process starts.

The program will work this way until the last game is played. All activities reflect the methods of the classes described above.

#### 2.1.4 The work of the source program.

To start working with the program in the program code, you need to change the value of the number of games to the desired one. It is also possible to load or unload the ANN weights into a file by commenting or uncommenting the lines. The program output results to the console window (Figure 2.5.).

```
Game #:2
Black player gets the first turn
Game over in 1159 moves
Num states: 579
black won
Point #0 has 0 pieces: []
Point #1 has 0 pieces: []
Point #2 has 0 pieces: []
Point #3 has 0 pieces: []
Point #4 has 0 pieces: []
Point #5 has 0 pieces: []
Point #6 has 0 pieces: []
Point #7 has 0 pieces: []
Point #8 has 0 pieces: []
Point #9 has 0 pieces: []
Point #10 has 0 pieces: []
Point #11 has 0 pieces: []
Point #12 has 0 pieces: []
Point #13 has 0 pieces: []
Point #14 has 0 pieces: []
Point #15 has 0 pieces: []
Point #16 has 0 pieces: []
Point #17 has 0 pieces: []
Point #18 has 0 pieces: []
Point #19 has 0 pieces: []
Point #20 has 0 pieces: []
Point #21 has 0 pieces: []
Point #22 has 2 pieces: ['white', 'white']
Point #23 has 11 pieces: ['white', 'white']
Win percentage: 0.0
Game #:3
```

Figure 2.5 – The output of the program

For each game is displayed:

- number;
- first player;
- total number of moves;
- number of board representations;

- winner;
- condition of the board at the time the game is completed;
- win percentage.

After the end of the last game, the program ends its execution.

### 2.1.5 Profiling the program.

To increase the efficiency of the program, it was decided to analyze the time costs of program functions. The program was profiled using the standard Python module cProfile. Profiling results are shown in Figure 2.6. They show that the getValue function has the worst ratio of time to the number of calls. It is responsible for processing the course of the game with an artificial neural network. Upon completion of the analysis, it became clear that productivity could be improved by introducing parallel computing into the program.

```

Ordered by: internal time
List reduced from 964 to 20 due to restriction <20>

```

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
33264478	153.388	0.000	153.388	0.000	D:\myGit\TD-Gammon-master\net.py:44(sigmoid)
3970	53.045	0.013	240.656	0.061	D:\myGit\TD-Gammon-master\net.py:47(getValue)
31954582	30.937	0.000	30.937	0.000	{method 'randn' of 'mtrand.RandomState' objects}
229	13.900	0.061	13.901	0.061	D:\myGit\TD-Gammon-master\net.py:87(do_td)
229	3.746	0.016	12.112	0.053	D:\myGit\TD-Gammon-master\net.py:74(feedforward)
3970	1.032	0.000	41.661	0.010	D:\myGit\TD-Gammon-master\net.py:56(<listcomp>)
1	0.469	0.469	268.018	268.018	train.py:1(<module>)
3745	0.368	0.000	0.499	0.000	D:\myGit\TD-Gammon-master\Game.py:273(get_representation)
3970	0.195	0.000	0.680	0.000	D:\myGit\TD-Gammon-master\net.py:58(<listcomp>)
757475	0.102	0.000	0.102	0.000	{method 'append' of 'list' objects}
517	0.055	0.000	0.069	0.000	D:\myGit\TD-Gammon-master\Game.py:41(find_moves)
3970	0.055	0.000	0.209	0.000	D:\myGit\TD-Gammon-master\net.py:54(<listcomp>)
396986/396954	0.050	0.000	0.050	0.000	{built-in method builtins.len}
565	0.040	0.000	0.040	0.000	{built-in method nt.stat}
30	0.033	0.001	0.033	0.001	{built-in method builtins.print}
121	0.022	0.000	0.022	0.000	{built-in method marshal.loads}
10	0.012	0.001	0.016	0.002	{built-in method _imp.create_dynamic}
3745	0.012	0.000	0.017	0.000	D:\myGit\TD-Gammon-master\Game.py:193(take_action)
326/324	0.010	0.000	0.017	0.000	{built-in method builtins.__build_class__}
7148	0.010	0.000	0.010	0.000	{method 'pop' of 'list' objects}

Figure 2.6 – The results of the profiling program

## 2.2 Training and analysis of ANN

To study the artificial neural network, the initial implementation has undergone changes. The original work processes were not changed, and the output of the configuration and intermediate calculations of the network was added to excel tables. It was trained on 8057 games. After each game, information about the state of the artificial

neural network was stored in a folder on the disk. After completing the training, the size of the data folder was 3,8 GB. The name of each file was named in accordance with the experience of the network at the time and the date the game was completed. The view of the files in the folder is shown in Figure 2.7.

Имя	Размер
inputs3583 : 02, Jan 2020, 00:23.xlsx	224 897
inputs3584 : 02, Jan 2020, 00:24.xlsx	269 738
inputs3585 : 02, Jan 2020, 00:26.xlsx	197 390
inputs3586 : 02, Jan 2020, 00:26.xlsx	327 147
inputs3587 : 02, Jan 2020, 00:28.xlsx	328 558
inputs3588 : 02, Jan 2020, 00:30.xlsx	370 790
inputs3589 : 02, Jan 2020, 00:33.xlsx	355 486
inputs3590 : 02, Jan 2020, 00:35.xlsx	357 128
inputs3591 : 02, Jan 2020, 00:37.xlsx	311 545
inputs3592 : 02, Jan 2020, 00:38.xlsx	405 542
inputs3593 : 02, Jan 2020, 00:41.xlsx	315 683
inputs3594 : 02, Jan 2020, 00:43.xlsx	342 505
inputs3595 : 02, Jan 2020, 00:45.xlsx	228 469
inputs3596 : 02, Jan 2020, 00:46.xlsx	276 924
inputs3597 : 02, Jan 2020, 00:47.xlsx	376 311
inputs3598 : 02, Jan 2020, 00:49.xlsx	282 832
inputs3599 : 02, Jan 2020, 00:51.xlsx	247 441
inputs3600 : 02, Jan 2020, 00:52.xlsx	300 533

Figure 2.7 – Folder with ANN data

In the files were written:

- hidden weight matrix 198x40;
- weight matrix of the output layer 40x2;
- input for the ANN for each move of 198 cells;
- matrix multiplication results;
- activation function results.

Due to the large amount of data, demonstrating tables in general and in a readable form is not possible.

In a study [28], it was revealed that some columns of the matrix of weights of the hidden layer contain only zeros or ones. In this regard, it was decided to conduct further analysis and comparison of the results of the ANN when sequentially truncating these columns, i.e., neurons.

## 2.3 Software design

To achieve the above goals, the need was identified for the implementation of a software product for monitoring and managing instances of the original ANN. For its implementation, design was carried out using SADT and UML.

### 2.3.1 Structural and functional model of the software product.

To describe the system as a whole, it was decided to use the SADT software design methodology. Using SADT, a structural analysis is carried out, which consists of two types of diagrams: an activity model and a data model. This methodology was released in 1981 by Douglas T. Ross under the name IDEF0.

After analyzing the source product, an «as is» model was created, as shown in Figure 2.8.

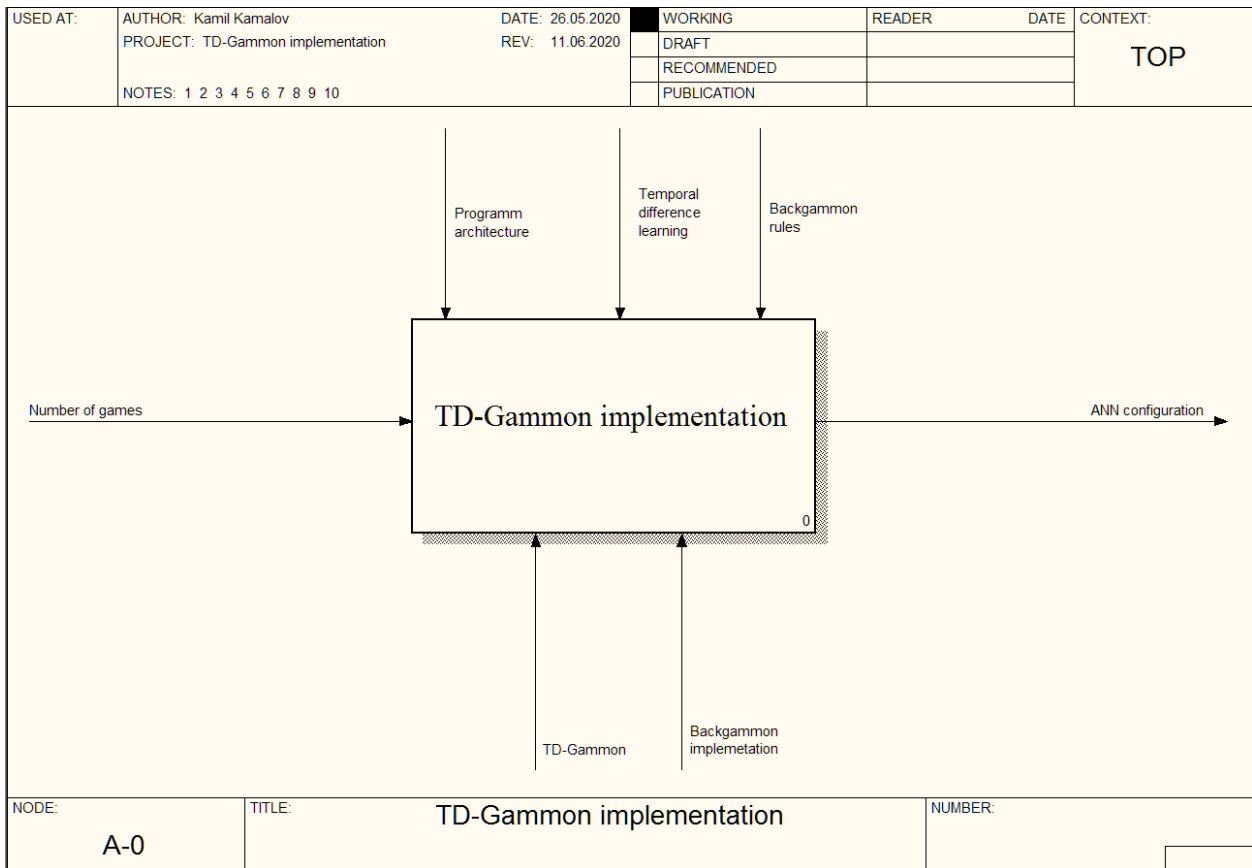


Figure 2.8 – Model «as is»

This model reflects the initial state of the system and helps to identify its shortcomings. So you can immediately notice the lack of interaction with the system, which makes it difficult to conduct experiments. This model was decomposed to identify internal work processes. It is shown in Figure 2.9.

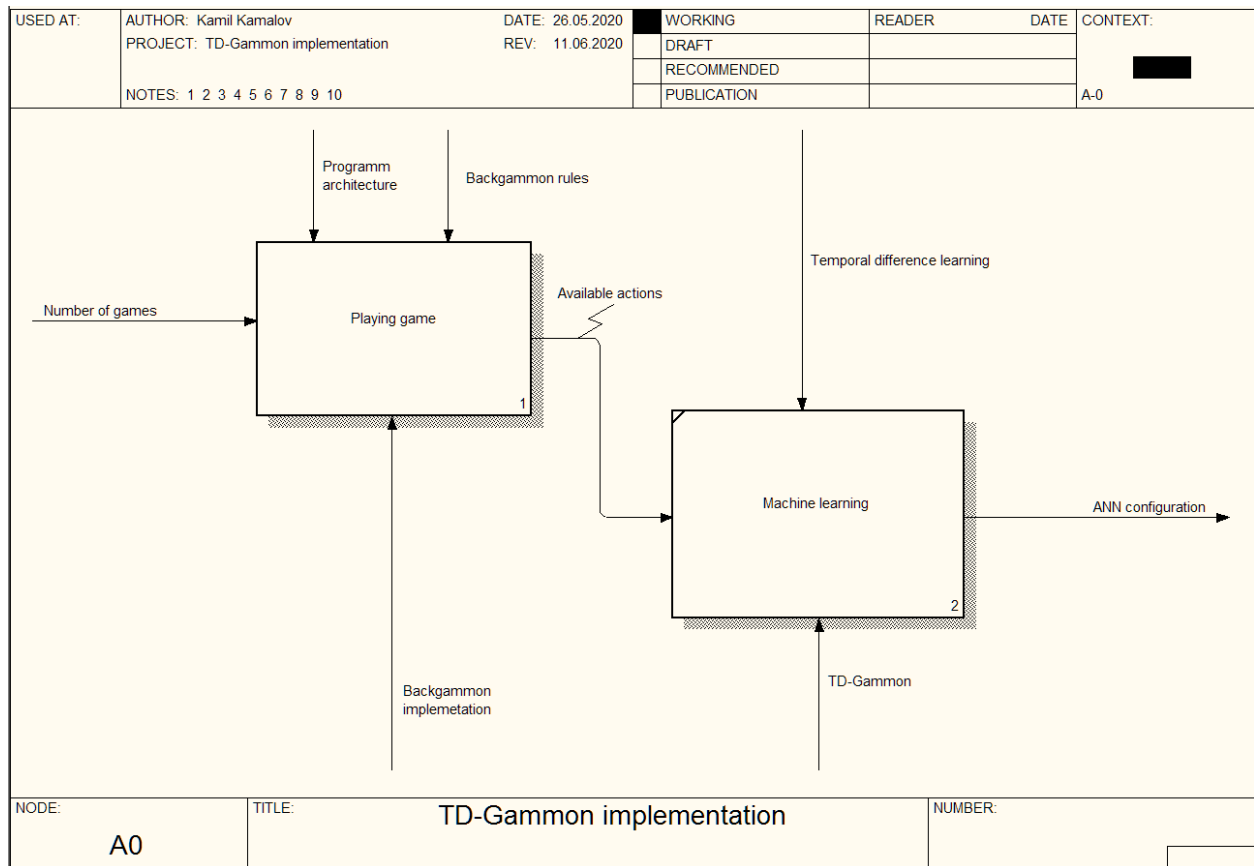


Figure 2.9 – Decomposition of the model «as is»

Next was built the model «to be». It improved the logic and structure of the model «as is». As you can see in Figure 2.10, the model was completely rethought, but the internal mechanisms remained the same to prevent incorrect results and qualitative analysis. So inputs were added for additional outputs. The implementation of new functions is monitored using the requirements specification. They will be described below.

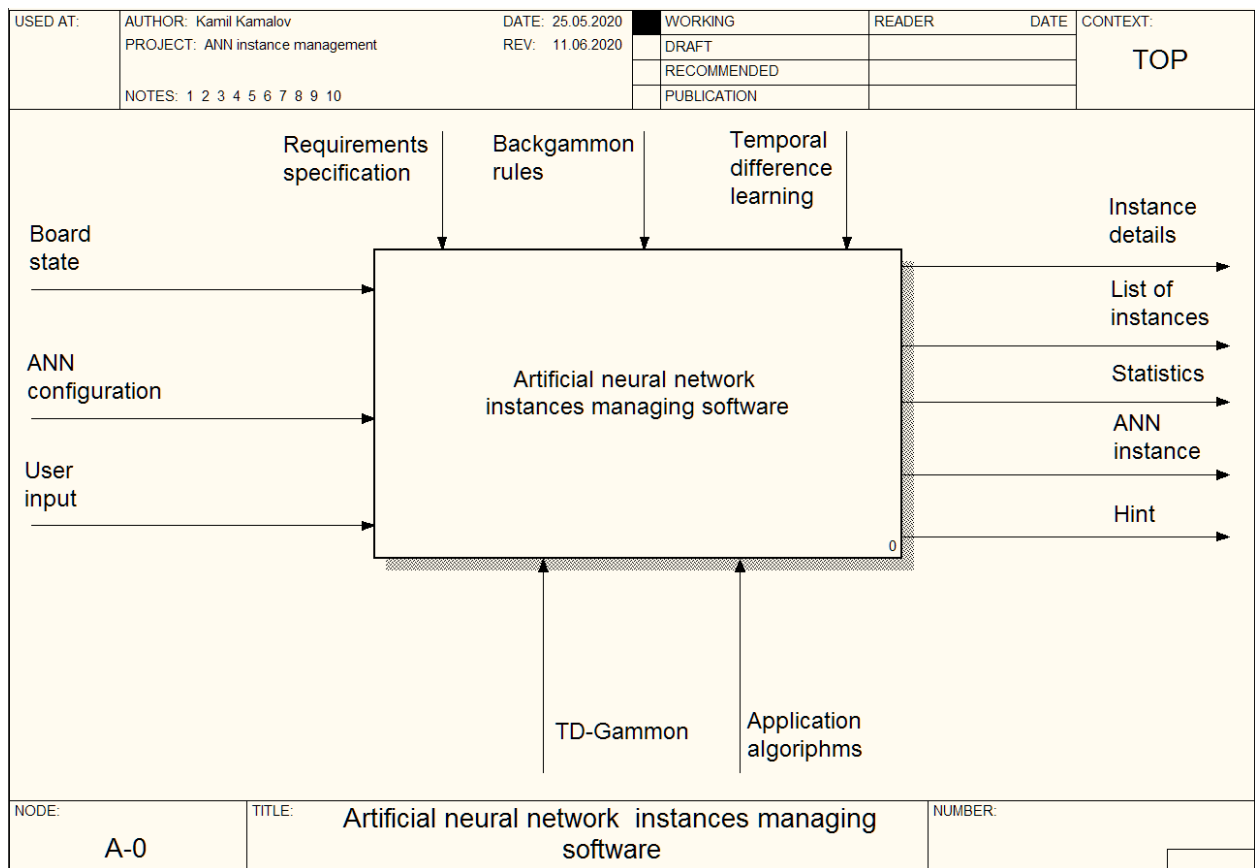


Figure 2.10 – Model «to be»

To identify additional functions, the model was decomposed (Figure 2.11). Four components describe the main functionality of the program and the data flow between them. Two additional modules appeared in the «to be» model. They are responsible for completing the assigned tasks.

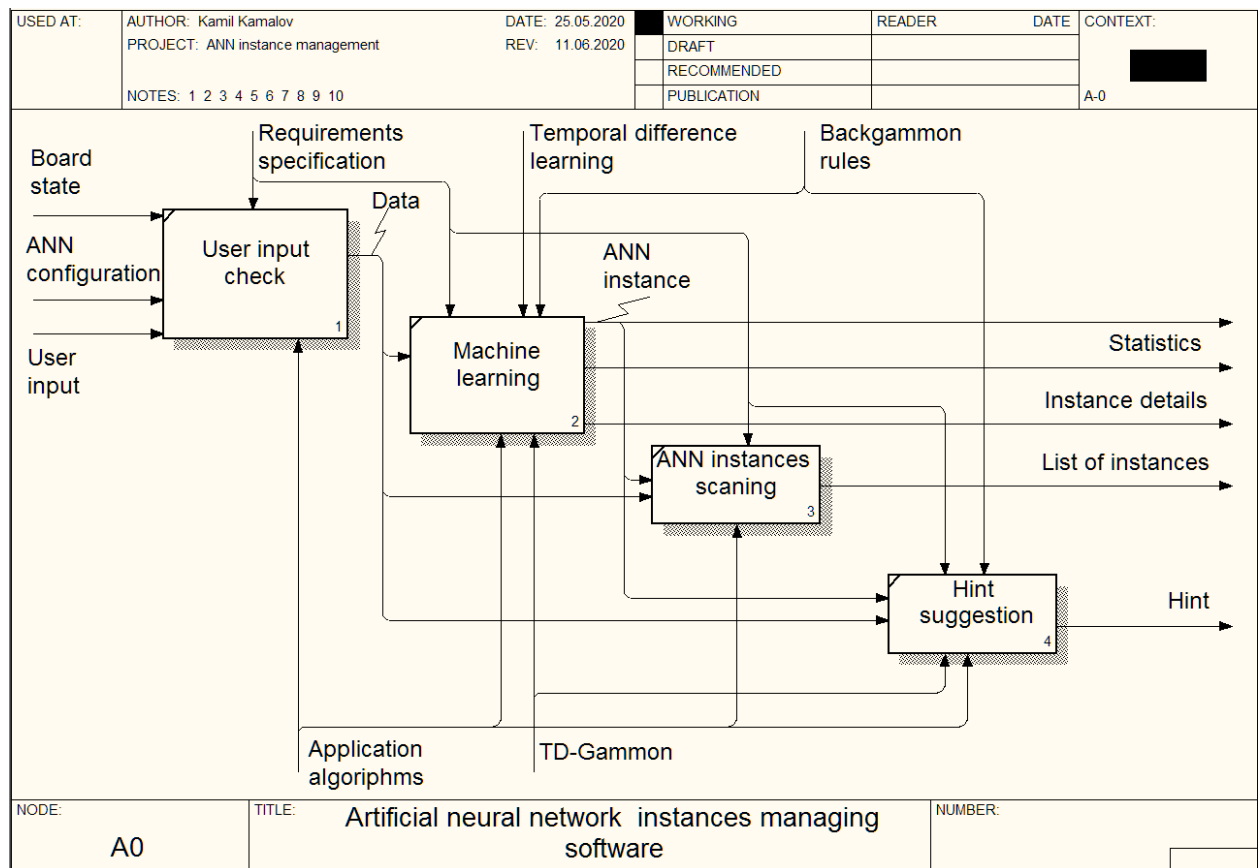


Figure 2.11 – Decomposition of the model «to be»

### 2.3.2 Analysis and detection of requirements.

In accordance with [29], a specification of requirements was created. It consists of tables of functions, as well as goals and objectives that were set earlier. The functions were divided into three groups: game functions, artificial neural network functions, and interface functions.

#### 2.3.2.1 Game functions.

The game functions describe the requirements for a module that implements the backgammon process. Only three functions will be available to the user for control. A complete list of game features is given in table 2.1.

Table 2.1 – List of game functions

Ref. #	Function	Category
--------	----------	----------



R1.1	Play backgammon	Evident
R1.2	Set opponents	Evident
R1.3	Read game state from example	Evident
R1.4	Roll dice	Hidden
R1.5	Find possible moves	Hidden
R1.6	Find the best move	Hidden
R1.7	Make a move	Hidden
R1.8	Undo move	Hidden
R1.9	Output game state	Hidden
R1.10	Find a winner	Hidden
R1.11	Identify opponent	Hidden
R1.12	Determine the condition of the board	Hidden
R1.13	Set board status	Hidden
R1.14	Display the contents of the point	Hidden

#### 2.3.2.2 Functions of the ANN.

The functions of an artificial neural network determine the requirements for its implementation module. For the user, most of the functions are available, since this group is responsible for carrying out the study and its tasks. The list of ANN functions is given in table 2.2.

Table 2.2 – Table of functions of the ANN

Ref. #	Function	Category
--------	----------	----------

R2.1	Make a prediction	Evident
R2.2	Learn	Evident
R2.3	Load weights	Evident
R2.4	Output weights	Evident
R2.5	Truncate weights	Evident
R2.6	Apply activation function	Hidden
R2.7	Run the TDL algorithm	Hidden
R2.8	Submit data for training	Hidden

#### 2.3.2.3 Interface functions.

Interface functions include functions responsible for the interaction of program modules. All explicit functions perform the tasks required to accomplish the purpose of the work. Using the interface functions, the user can interact with the rest of the program.

Table 2.3 – Interface functions table

Ref. #	Function	Category
R3.1	Create a new instance of ANN	Evident
R3.2	Continue training an existing instance of ANN	Evident
R3.3	Get a hint from an ANN instance	Evident
R3.4	Compare two instances of ANN	Evident
R3.5	Compare instances of ANN with the most experienced	Evident
R3.6	Check for files with ANN configurations	Evident

R3.7	Reconfigure ANN	Evident
R3.8	List existing instances	Hidden
R3.9	Display menu	Hidden

### 2.3.3 Use cases.

To achieve the goal of the work, four necessary use cases were identified. A diagram of use cases was constructed (Figure 2.12). To demonstrate their relationship with each other. Below is a detailed description of each of them along with the only identified actor.

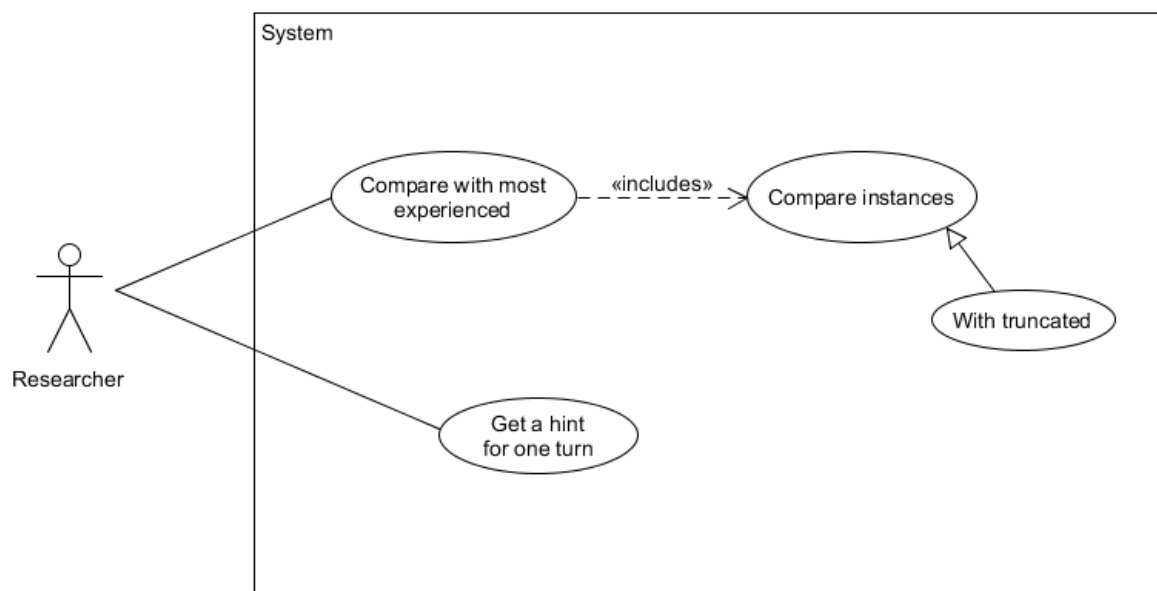


Figure 2.12 – UML use case diagram

#### 2.3.3.1 Actors.

During the analysis, only one role was discovered for using the system. This role is a researcher, and in particular an ANN researcher. He is the initiator of all processes, since each of them is necessary for the fulfillment of tasks in the framework of this final work.

### 2.3.3.2 Description of the use cases.

To understand the processes when using a software product, it is necessary to describe each of these cases. The description consists of the name of the case, its participants, the purpose for which it was initiated, the review and the functions used. The following are all cases of using the program.

Table 2.4 – Get a hint use case description

Use case:	<b>Getting hint for one move</b>
Actors:	Researcher (initiator)
Purpose:	Obtaining a prediction of the best action on the move to verify its adequacy
Overview:	The researcher in the main menu of the application selects the option to receive hints. The researcher selects an instance of the ANN from which he will receive a hint. The researcher enters the status of the board into a file and confirms the readiness. The researcher receives a prediction of the best move and gives him his assessment
Cross References:	R1.3, R1.4, R1.5, R1.6, R1.7, R1.8, R1.9, R1.13, R1.14, R2.1, R2.6, R3.3, R3.8, R3.9

Table 2.5 – Compare instances use case description

Use case:	<b>Comparison of the work of two instances of ANN</b>
Actors:	Researcher (initiator)
Purpose:	Getting statistics to compare the operation of different instances of ANN

Overview:	The researcher in the main menu of the application selects the option to compare two instances of the ANN. Researcher entering the number of games on which instances will be compared. The researcher selects two instances of the ANN for comparison. The researcher receives statistics and analyzes them
Cross References:	R1.1, R1.2, R1.4, R1.5, R1.6, R1.7, R1.8, R1.9, R1.10, R1.11, R1.12, R1.14, R2.1, R2.6, R3.4, R3.8, R3.9

Table 2.6 – Compare instances with truncated matrix use case description

Use case:	<b>Comparison of the work of two instances of ANN when truncating the weight matrix</b>
Actors:	Researcher (initiator)
Purpose:	Obtaining statistics for comparing the operation of ANN instances when truncating weights and without
Overview:	The researcher in the application menu selects the option to display the configuration of the ANN instance. The researcher in the application menu selects the option of truncating the weight matrix of the ANN instance. The researcher enters the instance number of the ANN and the column number of the weight matrix. The researcher in the main menu of the application selects the option to compare two instances of the ANN. Researcher entering the number of games on which instances will be compared. The researcher selects two instances of the ANN for comparison, one of which with truncated weights. The researcher receives statistics and analyzes them
Cross References:	R1.1, R1.2, R1.4, R1.5, R1.6, R1.7, R1.8, R1.9, R1.10, R1.11,

	R1.12, R1.14, R2.1, R2.3, R2.4, R2.6, R3.4, R3.7, R3.8, R3.9
--	--

Table 2.7 – Compare instances with most experienced use case description

Use case:	<b>Compare instances of ANN with the most experienced of them</b>
Actors:	Researcher (initiator)
Purpose:	Obtaining statistics to study changes in the operation of ANN instances
Overview:	The researcher in the main menu of the application selects the option of comparing ANN instances with the most experienced of them. Researcher receives statistics and analyzes them
Cross References:	R1.1, R1.2, R1.4, R1.5, R1.6, R1.7, R1.8, R1.9, R1.10, R1.11, R1.12, R1.14, R2.1, R2.6, R3.5, R3.8, R3.9

## 2.4 Design results

As design results, new class diagrams have been created. Existing classes were rethought without changing internal logic. A new interface class was designed that replaces and extends the functionality of the train.py file. The new file structure is shown in Figure 2.13.

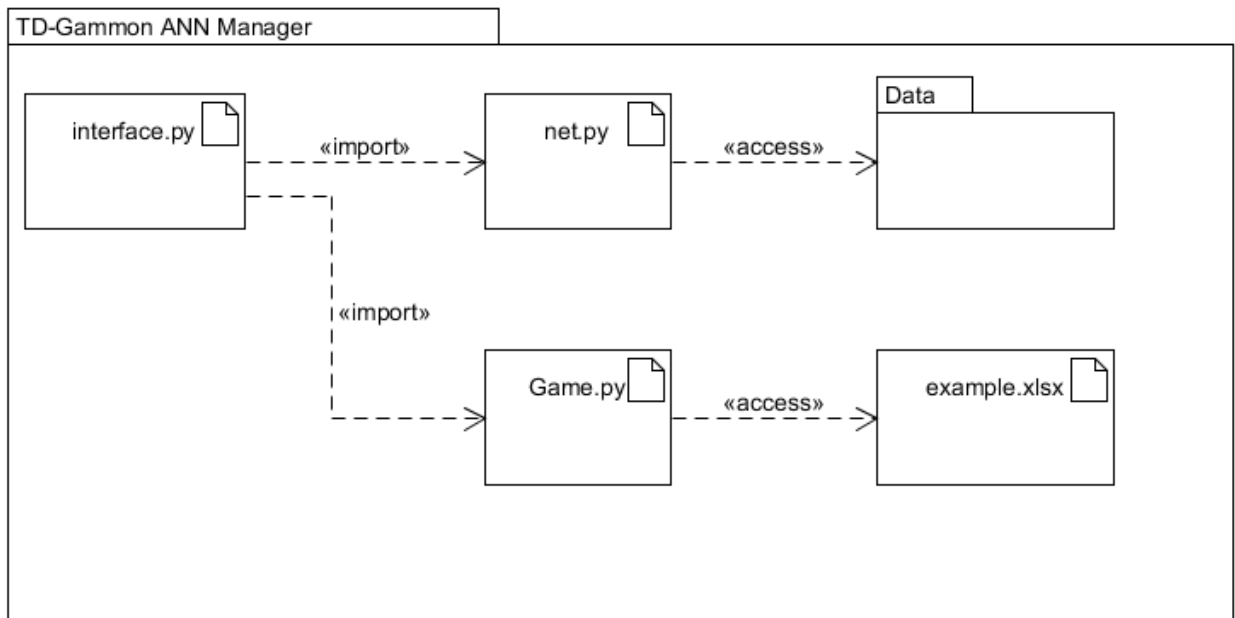


Figure 2.13 – UML diagram of packages of a new structure

#### 2.4.1 New Game.py artifact.

The function of the game process from the `train.py` file was added to this file. It was isolated and analyzed for subfunctions. The final view of the class model can be seen in Figure 2.14. New functionality was added in accordance with the specification of requirements, namely to get help on the entered move.

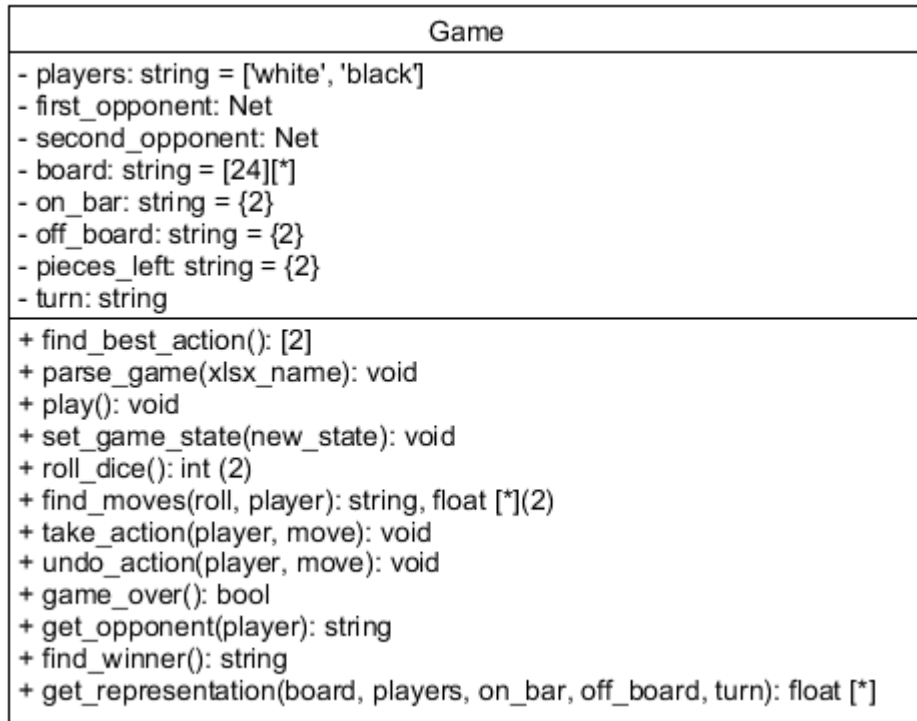


Figure 2.14 – UML diagram of the updated Game class

#### 2.4.2 New net.py artifact.

New properties have been added to the Net class to maintain information about each instance. Functionality was designed to change the configuration of the ANN in instances for testing. An updated view of the class is shown in Figure 2.15.



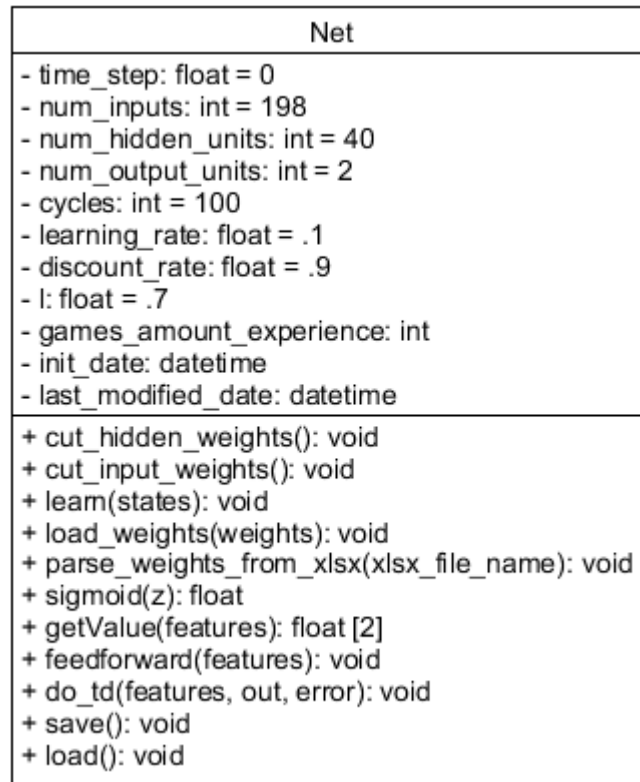


Figure 2.15 – UML diagram of the updated Net class

#### 2.4.3 New interface.py artifact.

The Interface class has been designed completely according to the requirements for performing the main tasks of the work. It contains all the functions necessary for conducting tests. This file completely replaces the train.py file and provides the user with the opportunity to interact with the program. This class was presented as an interface in Figure 2.16.

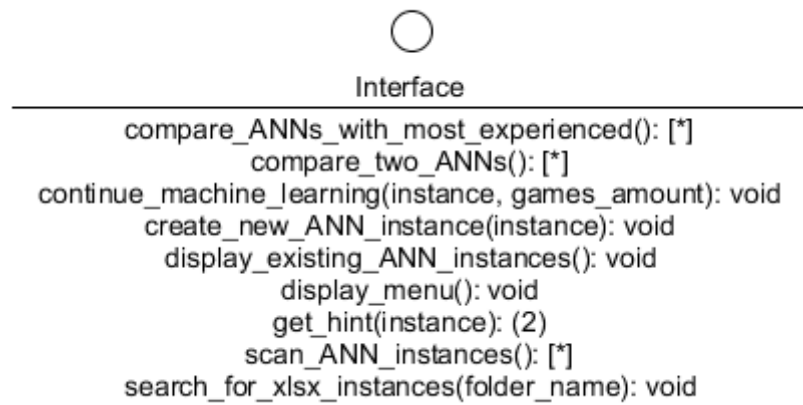


Figure 2.16 – UML diagram of the Interface class

### **3 Software implementation, testing and obtaining results**

#### **3.1 Selection of development tools**

Python was used as a programming language. This choice is justified by the fact that the initial implementation of the program was written in this language. Python is an interpreted programming language that has many advantages. Program code in this language is easy to write and read. Many people prefer Python precisely because of its simplicity.

The program was written in a text editor Vim. Vim has a great history and uniqueness. It is very different from the usual text editors or development environments. The emphasis in it is on manipulating the text, and not on its set. Its drawback is the need for prior use training. At the same time, understanding his work helps significantly reduce the time it takes to write programs.

Git version control was also used. Its use was necessary for the development process. Often there was a need to view earlier versions and sometimes rollback to them.

All UML models were designed using the UMLet tool. Unlike other analogues, this software is freely distributed and at the same time has wide functionality. Absolutely all elements can be changed to fit your needs.

The development process was monitored using the Trello service. This product has been used because of its simplicity. Access to it can be obtained at any time via the Internet, and its content is intuitive to use.

#### **3.2 Description of the structure and operation of the program**

The program starts with the main menu cycle. It contains the main functions of the program, which can be seen in Figure 3.1. Exiting the program is provided by pressing the «x» key.

```
Backgammon game using TD-Gammon
Available options:
  1. Make a new instance of ANN
  2. Continue process of machine learning
  3. Get hint for one turn
  4. Compare two ANNs
  5. Compare all ANNs to most experienced
  6. Search for old examples
  7. Configure instance of ANN

Enter option number to continue
Press X to exit

Enter option number:
```

Figure 3.1 – The main menu of the program

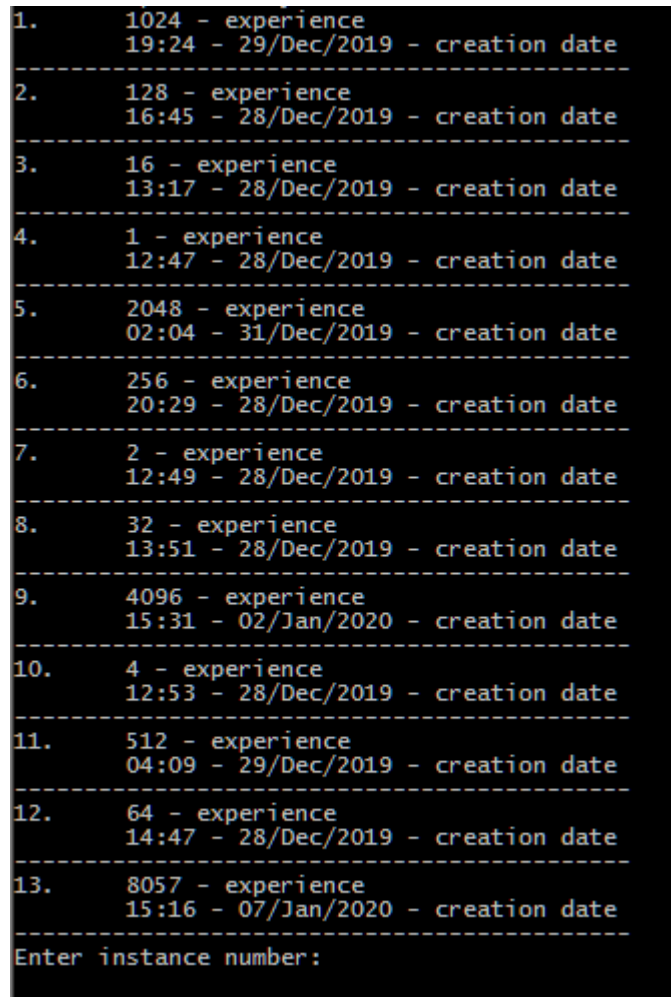
When an option is selected, control of the specified function is transferred. When you select the first option to create a new instance of the ANN, an object of the Net class is initialized and you are asked to enter the number of games for training (Figure 3.2).

```
Point #20 has 0 pieces: []
Point #21 has 0 pieces: []
Point #22 has 0 pieces: []
Point #23 has 6 pieces: ['white', 'white', 'white', 'white', 'white', 'white']
Game #:5
2020-06-08 18:52:57.861219
Black player gets the first turn
Game over in 429 moves
Num states: 214
black won
Point #0 has 0 pieces: []
Point #1 has 0 pieces: []
Point #2 has 0 pieces: []
Point #3 has 0 pieces: []
Point #4 has 0 pieces: []
Point #5 has 0 pieces: []
Point #6 has 0 pieces: []
Point #7 has 0 pieces: []
Point #8 has 0 pieces: []
Point #9 has 0 pieces: []
Point #10 has 0 pieces: []
Point #11 has 0 pieces: []
Point #12 has 0 pieces: []
Point #13 has 0 pieces: []
Point #14 has 0 pieces: []
Point #15 has 0 pieces: []
Point #16 has 0 pieces: []
Point #17 has 0 pieces: []
Point #18 has 0 pieces: []
Point #19 has 0 pieces: []
Point #20 has 0 pieces: []
Point #21 has 0 pieces: []
Point #22 has 0 pieces: []
Point #23 has 11 pieces: ['white', 'white', 'white', 'white', 'white', 'white', 'white', 'white', 'white', 'white', 'white']
```

Figure 3.2 – Selecting the option to create a new instance

The first option implicitly includes the second. This was implemented in this way to reduce the amount of code and avoid repetition. Logically, this is justified by the fact that the new instance also continues to learn.

Options 2, 3, 4, 7 have the same initial output (Figure 3.3) since their operation requires the selection of an ANN instance.



1.	1024 - experience	19:24 - 29/Dec/2019 - creation date
2.	128 - experience	16:45 - 28/Dec/2019 - creation date
3.	16 - experience	13:17 - 28/Dec/2019 - creation date
4.	1 - experience	12:47 - 28/Dec/2019 - creation date
5.	2048 - experience	02:04 - 31/Dec/2019 - creation date
6.	256 - experience	20:29 - 28/Dec/2019 - creation date
7.	2 - experience	12:49 - 28/Dec/2019 - creation date
8.	32 - experience	13:51 - 28/Dec/2019 - creation date
9.	4096 - experience	15:31 - 02/Jan/2020 - creation date
10.	4 - experience	12:53 - 28/Dec/2019 - creation date
11.	512 - experience	04:09 - 29/Dec/2019 - creation date
12.	64 - experience	14:47 - 28/Dec/2019 - creation date
13.	8057 - experience	15:16 - 07/Jan/2020 - creation date
Enter instance number:		

Figure 3.3 – Selection of the ANN instance

When choosing the third option, you must confirm that the file with the example board is prepared. It is an Excel table (Figure 3.4) in which checkers of black color are encoded with the number 0 and white 1. The playing field is in the range A1: P20, where the gray cells indicate the bar, and each column between them has two opposite points. The turn order is recorded in cell H23, and the throw in I23. Interaction with Excel tables in Python is implemented using the third-party Openpyxl library. When the function is completed, the proposed move is displayed on the screen (Figure 3.5).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		24	23	22	21	20	19			18	17	16	15	14	13	
2		0														
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																
17																
18		1														
19		1														
20		1	2	3	4	5	6			7	8	9	10	11	12	
21																
22								Ход	Бросок							
23								0	(1, 2)							

Figure 3.4 – An example of filling the state of the game

```
Enter instance number: 13
Next move - (24, 22)
Press any key to continue...
```

Figure 3.5 – The predicted move of the program

Options 4 and 5 have the same output. Option 5 contains option 4, since the fifth option also compares two instances of the neural network, but in one cycle. The results of these functions are data that consists of the percentage of winning and the number of moves spent. Charts in the program are generated automatically by using the matplotlib charting package. An example of the graph is shown in Figure 3.6. The graph is a solid line since a test was performed where all instances played against the most experienced among them. As a result, the percentage of victories did not increase.

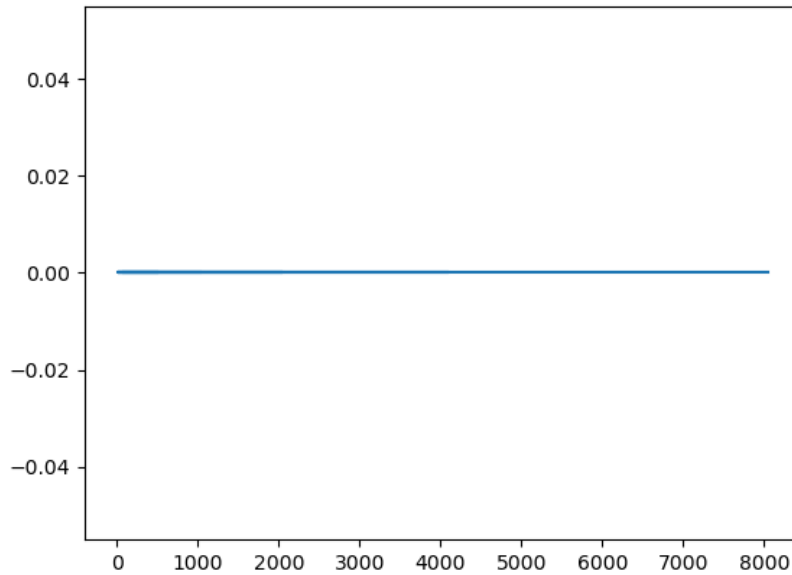


Figure 3.6 – Example program schedule

The sixth option was implemented to read old-format scales. For it to work, you must enter the path to the folder in which the search for scales in files will take place.

The seventh option takes on the input the column number of the weight matrix to truncate it. To implement this function, there was no need to change the original program code. When truncating one of the neurons, the calculations are correct, since they are not tied to the size of the original matrix. This option has no output, ANN with truncated weights is saved as a new instance and is ready for use.

The program reproduced parallel computations using the internal multiprocessing library. A repeating piece of code was parallelized to handle each of the possible moves. This optimization allowed to significantly reduce the operating time, but depending on the processor used and the number of its cores.

### 3.3 Results of the work

According to the results of the tests, it became clear that the trained network at all stages gave preference to the player playing black drafts. As shown in the Figure 3.6 the instance with the most experience has not lost a single game. Even when playing with

himself, black always won. Testing was carried out on all copies of 10 games each. Testing on more games was not possible due to the huge computational time.

From this we can conclude that the percentage of gain cannot be a reliable criterion for evaluating the results of the operation of an artificial neural network. However, according to the research, it was found that the forecast indicated by the author came true. It was found that with an increase in the experience of the neural network, the number of moves decreased, which indicates the adoption of its more reasonable decisions. In this regard, it was decided to determine by the criterion for evaluating the results the number of moves of the pieces by the neural network during the match. A graph of the number of completed moves is shown in Figure 3.7 and you can see a downward trend on it.

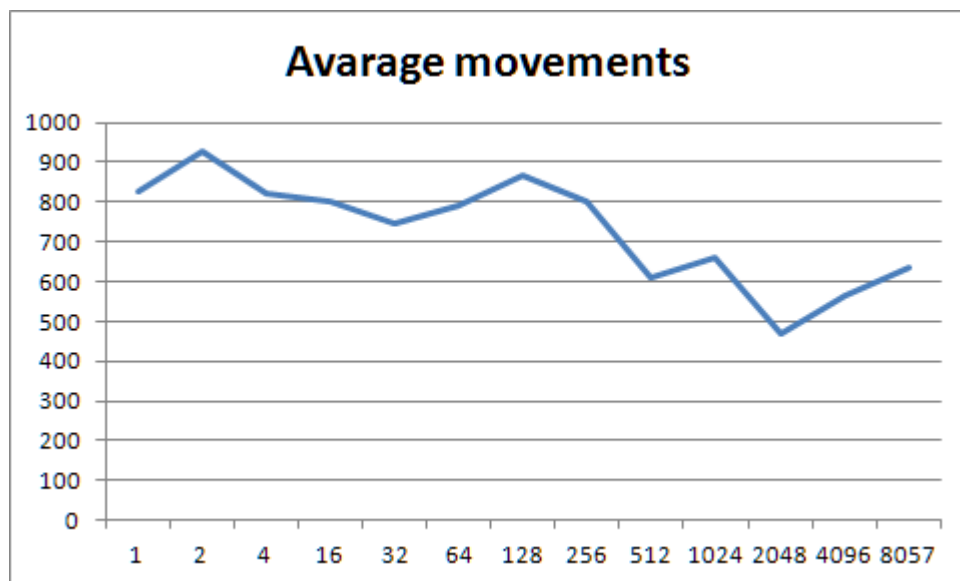


Figure 3.7 – Graph of the average number of movements

Comparison of the results of work with truncation of the matrix of weights did not give a deterioration. Testing the most experienced specimen with truncating weights and without in ten games showed an average result of 688 moves, which proves the correctness of the assumptions of [28].



## CONCLUSION

During the final qualification work, an analysis of the subject area was carried out, in which the basic concepts of artificial neural networks, the principles of backgammon, and the methodology with which it was designed, were explained.

A context model was built and its decomposition according to the IDEF0 standard. The initial product was depicted as an «as is» model, and the final «to be» model. An object analysis of the task was performed and a specification of requirements was drawn up. On its basis, use cases were painted. All diagrams except the context one were executed in UML.

On this basis, a software product for managing instances of an artificial neural network was implemented. It provides the ability to create and edit network instances to conduct tests on them. Also, its functionality provides opportunities to achieve the goal of the work.

According to the results of testing the program, it became clear that the initially selected evaluation criterion is not suitable for comparisons. In the course of research and analysis, a new one was discovered that clearly shows the network's training.

Based on the results obtained, we can say that the goal and objectives of the final qualification work are completed. To conduct further research, it is expected to continue the training of an artificial neural network and its reengineering.

## REFERENCES

- 1 Shanmuganathan, S. Artificial neural network modelling: An introduction / S. Shanmuganathan // Artificial neural network modelling. – Springer, Cham, 2016. – Vol.628. – P. 1-14.
- 2 Nikulin, Y. Exploring the neural algorithm of artistic style / Y. Nikulin, R. Novak // arXiv preprint arXiv:1602.07188. – Vol. 47. – 2016.
- 3 Krenker, A. Introduction to the artificial neural networks / A. Krenker, J. Bešter, A. Kos // Artificial Neural Networks: Methodological Advances and Biomedical Applications. – IntechOpen, 2011. – P. 1-18.
- 4 Shanthi, D. Evolving connection weights of artificial neural networks using genetic algorithm with application to the prediction of stroke disease / D. Shanthi, G. Sahoo, N. Saravanan // International Journal of Soft Computing. – 2009. – Vol. 4. – №. 2. – P. 95-102.
- 5 Однослойные искусственные нейронные сети [Electronic resource]. – URL: <https://www.intuit.ru/studies/courses/88/88/lecture/20527?page=3> (date of the application: 12.05.2020).
- 6 Пишем свою нейросеть: пошаговое руководство [Electronic resource]. – URL: <https://proglib.io/p/neural-nets-guide> (date of the application: 12.05.2020).
- 7 Редукция нейронных сетей при помощи вариационной оптимизации [Electronic resource]. – URL: <https://habr.com/ru/post/413939/> (date of the application: 12.05.2020).
- 8 Feng, J. Performance Analysis of Various Activation Functions in Artificial Neural Networks / J. Feng, S. Lu // Journal of Physics: Conference Series. – IOP Publishing, 2019. – Vol. 1237. – №. 2. – P. 022030.
- 9 Sibi, P. Analysis of different activation functions using back propagation neural networks / P. Sibi, S. A. Jones, P. Siddarth // Journal of Theoretical and Applied Information Technology. – 2013. – Vol. 47. – №. 3. – P. 1264-1268.
- 10 Livingstone, D.J. (ed.). Artificial neural networks: methods and applications. / D.J. Livingstone – Totowa, NJ: Humana Press, 2008. – P. 185-202.

- 11 Han, S.H. Artificial neural network: understanding the basic concepts without mathematics / S.H. Han, K.W. Kim, S. Kim, Y.C. Youn // Dementia and neurocognitive disorders. – 2018. – Vol. 17. – №. 3. – P. 83-89.
- 12 Gradient Descent and Stochastic Gradient Descent [Electronic resource]. – URL: [https://rasbt.github.io/mlxtend/user\\_guide/general\\_concepts/gradient-optimization/](https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/) (date of the application: 26.05.2020).
- 13 Abiodun, O.I. State-of-the-art in artificial neural network applications: A survey / O.I. Abiodun, A. Jantan, A.E. Omolara et al. // Heliyon. – 2018. – Vol. 4. – №. 11. – P. e00938.
- 14 Young, W.A. Using a heuristic approach to derive a grey-box model through an artificial neural network knowledge extraction technique / W.A. Young, G.R. Weckman // Neural computing and applications. – 2010. – Vol. 19. – №. 3. – P. 353-366.
- 15 A History of Backgammon [Electronic resource]. – URL: <https://www.bkgm.com/articles/GOL/Nov00/mark.htm> (date of the application: 27.04.2020).
- 16 Nackgammon [Electronic resource]. – URL: <http://www.nackbg.com/nackgammon/> (date of the application: 28.04.2020).
- 17 LongGammon [Electronic resource]. – URL: <http://www.bkgm.com/variants/LongGammon.html> (date of the application: 28.04.2020).
- 18 Hyper-backgammon [Electronic resource]. – URL: <http://www.bkgm.com/variants/HyperBackgammon.html> (date of the application: 28.04.2020).
- 19 Rules of Backgammon [Electronic resource]. – URL: <https://www.bkgm.com/rules.html> (date of the application: 28.04.2020).
- 20 Tesauro, G. TD-Gammon, a self-teaching backgammon program, achieves master-level play / G. Tesauro // Neural computation. – 1994. – Vol. 6. – №. 2. – P. 215-219.
- 21 Berliner, H.J. Backgammon computer program beats world champion / H.J. Berliner // Artificial Intelligence. – 1980. – Vol. 14. – №. 2. – P. 205-220.
- 22 Tesauro, G. Temporal Difference Learning and TD-Gammon / G. Tesauro // Communications of the ACM. – 1995. – Vol. 38. – №. 3. – P. 58-68.

- 23 Pollack, J.B. Why did TD-gammon work? / J.B. Pollack, A.D. Blair // Advances in Neural Information Processing Systems. – 1997. – P. 10-16.
- 24 FIPS 183. Announcing the Standard for integration definition for function modeling (IDEF0) [Electronic resource]. – URL: <http://www.iso.staratel.com/IDEF/IDEF0/IDEF0.pdf> (date of the application: 28.04.2020).
- 25 Фаулер, М. UML. Основы, 3-е издание / М. Фаулер. Перевод с английского А. Петухова. – СПб.: Символ-Плюс, 2004. – 192 с.
- 26 Liu, Z. Object-oriented software development with UML / Z. Liu. – Technical Report UNU/IIST Report, 2002. – №. 259. – 179 p.
- 27 TD-Gammon [Electronic resource]. – URL: <https://github.com/millerm/TD-Gammon> (date of the application: 28.11.2019).
- 28 Муфтахов, Т.И. Замена функции активации на пороговую на примере TD-Gammon / Т.И. Муфтахов, В.М. Гиниятуллин, К.И. Камалов, Д.А. Чурилов, М.А. Салихова // Актуальные проблемы науки и техники – 2020: материалы XIII Междунар. конф. молодых ученых / УГНТУ. – Уфа, 2020. – Т. 2. – С. 134-137.

## **Appendix A**

(obligatory)

### **List of illustrative and graphic materials of the GQW**

#### **List of demonstration materials of the GQW**

- 1 Presentation material (presentation)
- 2 Demonstration of the computerized system (video clip)

#### **GQW's list of figures**

	Page
Figure 1.1 – Comparison of a biological and artificial neuron .....	9
Figure 1.2 – Schematic representation of an artificial neuron .....	11
Figure 1.3 – An example of using a weight matrix .....	11
Figure 1.4 – The process of removing a neuron.....	12
Figure 1.5 – Sigmoid curve and its formula.....	14
Figure 1.6 – A graph of the cost function .....	15
Figure 1.7 – Classification of artificial neural networks .....	15
Figure 1.8 – Feedforward network with two layers of neurons .....	16
Figure 1.9 – Feed-backward neural network.....	16
Figure 1.10 – Visualization of concepts black, white, gray boxes.....	17
Figure 1.11 – Board for playing backgammon .....	18
Figure 1.12 – Nackgammon .....	19
Figure 1.13 – Longgammon .....	19
Figure 1.14 – Hyper-backgammon .....	19
Figure 1.15 – Board with checkers in the initial positions .....	20
Figure 1.16 – Doubling cube .....	21
Figure 1.17 – Two variants of movements .....	22

Figure 1.18 – The white player throws out 6–4 .....	22
Figure 1.19 – White player throws checkers.....	23
Figure 1.20 – Multilayer perceptron used in TD-Gammon.....	25
Figure 1.21 – Box in diagram.....	27
Figure 1.22 – Types of arrows in the model .....	28
Figure 1.23 – Standard positions of arrows in the diagram and their roles.....	29
Figure 1.24 – An example of a top-level diagram.....	30
Figure 1.25 – Structure of decomposition.....	31
Figure 1.26 – The requirements analysis process.....	32
Figure 1.27 – Classification of UML diagrams.....	34
Figure 1.28 – Structural things in UML.....	35
Figure 1.29 – Behavioral things in UML.....	35
Figure 1.30 – Grouping and annotation things in UML .....	36
Figure 1.31 – Relations in UML.....	36
Figure 2.1 – UML diagram of initial implementation packages.....	37
Figure 2.2 – UML diagram of the Net class.....	38
Figure 2.3 – UML Game class diagram.....	39
Figure 2.4 – UML activity diagram of the train.py file .....	41
Figure 2.5 – The output of the program.....	42
Figure 2.6 – The results of the profiling program .....	43
Figure 2.7 – Folder with ANN data .....	44
Figure 2.8 – Model «as is» .....	45
Figure 2.9 – Decomposition of the model «as is».....	46
Figure 2.10 – Model «to be» .....	47
Figure 2.11 – Decomposition of the model «to be» .....	48
Figure 2.12 – UML use case diagram.....	51
Figure 2.13 – UML diagram of packages of a new structure .....	55
Figure 2.14 – UML diagram of the updated Game class.....	56
Figure 2.15 – UML diagram of the updated Net class .....	57
Figure 2.16 – UML diagram of the Interface class .....	58

Figure 3.1 – The main menu of the program .....	60
Figure 3.2 – Selecting the option to create a new instance.....	60
Figure 3.3 – Selection of the ANN instance .....	61
Figure 3.4 – An example of filling the state of the game .....	62
Figure 3.5 – The predicted move of the program.....	62
Figure 3.6 – Example program schedule .....	63
Figure 3.7 – Graph of the average number of movements .....	64

### **GQW's list of tables**

	Page
Table 2.1 – List of game functions .....	48
Table 2.2 – Table of functions of the ANN .....	49
Table 2.3 – Interface functions table .....	50
Table 2.4 – Get a hint use case description.....	52
Table 2.6 – Compare instances with truncated matrix use case description .....	53
Table 2.7 – Compare instances with most experienced use case description .....	54

## **Appendix B**

(recommended)

### **Programmer's manual**

#### **B.1 Program purpose**

This system is designed to conduct research on the software implementation of the ANN TD-Gammon. The main goal of creating this software is to compare the results of an artificial neural network with sequential truncation of the weight matrix.

#### **B.2 Purpose and conditions of use**

The program works in console mode. The system receives input from there and outputs the results there. Almost all program options are extremely time consuming. This is due to the large number of calculations necessary for the operation of the ANN. The number of calls and their execution time is shown in Figure B.1. It is also worth noting that the operating time of the program depends on the processor power and the number of its cores, since during the development optimizations were made in the form of introducing parallel computing. A standard Python multiprocessing library was used to apply parallel computing. An example usage code is shown in Figure B.2.



Ordered by: internal time  
List reduced from 964 to 20 due to restriction <20>

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
33264478	153.388	0.000	153.388	0.000	D:\myGit\TD-Gammon-master\net.py:44(sigmoid)
3970	53.045	0.013	240.656	0.061	D:\myGit\TD-Gammon-master\net.py:47(getValue)
31954582	30.937	0.000	30.937	0.000	{method 'randn' of 'mtrand.RandomState' objects}
229	13.900	0.061	13.901	0.061	D:\myGit\TD-Gammon-master\net.py:87(do_td)
229	3.746	0.016	12.112	0.053	D:\myGit\TD-Gammon-master\net.py:74(feedforward)
3970	1.032	0.000	41.661	0.010	D:\myGit\TD-Gammon-master\net.py:56(<listcomp>)
1	0.469	0.469	268.018	268.018	train.py:1(<module>)
3745	0.368	0.000	0.499	0.000	D:\myGit\TD-Gammon-master\Game.py:273(get_representation)
3970	0.195	0.000	0.680	0.000	D:\myGit\TD-Gammon-master\net.py:58(<listcomp>)
757475	0.102	0.000	0.102	0.000	{method 'append' of 'list' objects}
517	0.055	0.000	0.069	0.000	D:\myGit\TD-Gammon-master\Game.py:41(find_moves)
3970	0.055	0.000	0.209	0.000	D:\myGit\TD-Gammon-master\net.py:54(<listcomp>)
896986/396954	0.050	0.000	0.050	0.000	{built-in method builtins.len}
565	0.040	0.000	0.040	0.000	{built-in method nt.stat}
30	0.033	0.001	0.033	0.001	{built-in method builtins.print}
121	0.022	0.000	0.022	0.000	{built-in method marshal.loads}
10	0.012	0.001	0.016	0.002	{built-in method _imp.create_dynamic}
3745	0.012	0.000	0.017	0.000	D:\myGit\TD-Gammon-master\Game.py:193(take_action)
326/324	0.010	0.000	0.017	0.000	{built-in method builtins.__build_class__}
7148	0.010	0.000	0.010	0.000	{method 'pop' of 'list' objects}

Figure B.1 – The results of the profiling program

```

with Pool() as pool:
    try:
        if self.turn == 'white':
            values = pool.map(self.first_opponent.getValue, repr_list)
        elif self.turn == 'black':
            values = pool.map(self.second_opponent.getValue, repr_list)
    except KeyboardInterrupt:
        pool.terminate()
        print("Game is terminated")
        return

```

Figure B.2 – Usage of multiprocessing

Since the development was carried out using an object-oriented programming paradigm, understanding and further editing the program should not cause difficulties. The menu code for the application where each of the methods are called is shown in Figure B.3.

```

# Forming list of all options
self.options = [
    "Make a new instance of ANN",
    "Continue process of machine learning",
    "Get hint for one turn",
    "Compare two ANNs",
    "Compare all ANNs to most experienced",
    "Search for old examples",
    "Configure instance of ANN",
]

# Main control cycle of programm
while True:

    self.ANN_instances = self.scan_ANN_instances()
    if len(self.ANN_instances) > 1:
        self.more_than_one_network_instance = True
    else:
        self.more_than_one_network_instance = False

    os.system('cls')
    self.display_menu()
    choosen_option = input("\nEnter option number: ")
    print()

    if choosen_option == 'x':
        break

    elif choosen_option == '1':
        os.system('cls')
        games_amount = input("Enter desired experience of ANN"
                              "(games amount): ")
        games_amount = int(games_amount)
        self.create_new_ANN_instance(games_amount)

    elif choosen_option == '2':
        os.system('cls')
        print("Choose what ANN you want to continue to train:\n")
        self.display_existing_ANN_instances()

        instances = self.scan_ANN_instances()
        choosen_instance_number = int(input("Enter instance number: "))
        choosen_instance_name = instances[choosen_instance_number-1]
        with open(choosen_instance_name, 'rb') as pi:
            ANN_instance = pickle.load(pi)
        games_amount = input("Enter desired additional experience of ANN"
                              "(games amount): ")
        games_amount = int(games_amount)
        self.continue_machine_learning(ANN_instance, games_amount)

    elif choosen_option == '3':
        os.system('cls')
        answer = input("Enter 'y' is you made your setup in example.xlsx.\n"
                      "Otherwise press 'n'.")
        if answer == 'n':
            continue

```

Figure B.3 – Menu code

The entire code contains comments in the form of comments and documentation is written for each class method. You can see the documentation of methods in Python in Figure B.4.

```
def compare_two_ANNs(self, first_instance, second_instance, games_amount):
    """
    Two given instances of ANN will play against each other for
    entered amount of games. Result is win percentage for first
    chosen network.
    """
```

Figure B.4 – Doc strings example

To run the program, you must provide an input file with the interface.py code in the Python interpreter (Figure B.5).

```
D:\myGit\TD-Gammon>python interface.py
```

Figure B.5 – Launch program from console

### B.3 Input and Output

The input and output files for the program are serialized instances of the Net class. Data is serialized using the Python pickle standard library. Files are saved with names including modification dates and experience of instances. Data is stored in the data directory (Figure B.6). File names include change dates and experience of instances. Code snippets for saving and loading instances can be seen in Figures B.7 and B.8.

Имя ^	Дата изменения	Тип	Размер
1_03_Jun_2020__h12_m56.pkl	03.06.2020 15:12	Файл "PCL"	966 КБ
1_21_May_2020__h22_m21.pkl	21.05.2020 22:21	Файл "PCL"	342 КБ
1_22_May_2020__h11_m58.pkl	22.05.2020 11:58	Файл "PCL"	342 КБ
1_22_May_2020__h12_m09.pkl	22.05.2020 12:09	Файл "PCL"	342 КБ
1_22_May_2020__h23_m03.pkl	22.05.2020 23:12	Файл "PCL"	966 КБ
1_23_May_2020__h15_m24.pkl	23.05.2020 16:34	Файл "PCL"	966 КБ
1_26_May_2020__h00_m19.pkl	26.05.2020 1:31	Файл "PCL"	966 КБ
2_21_May_2020__h22_m41.pkl	21.05.2020 22:41	Файл "PCL"	966 КБ
3_23_May_2020__h00_m31.pkl	23.05.2020 0:41	Файл "PCL"	966 КБ
4_22_May_2020__h13_m08.pkl	22.05.2020 13:08	Файл "PCL"	966 КБ
6_22_May_2020__h22_m09.pkl	22.05.2020 22:09	Файл "PCL"	966 КБ

Figure B.6 – Serialized instances in data folder

```
def save(self):
    self.last_modified_date = datetime.now()
    with open('data/{}_{}_{:d}_{:b}_{:Y}_{:h}_{:M}_{:M}.pkl'
              .format(self.games_amount_experience, self.init_date)
              , 'wb') as pickled_instance:
        pickle.dump(self, pickled_instance)
```

Figure B.7 – Save instance method

```
self.display_existing_ANN_instances()
instances = self.scan_ANN_instances()
chosen_instance_number = int(input("Enter instance number: "))
chosen_instance_name = instances[chosen_instance_number-1]
with open(chosen_instance_name, 'rb') as pi:
    ANN_instance = pickle.load(pi)
```

Figure B.8 – Load particular instance

#### B.4 Messages of the system

All program and input errors are handled by the Python interpreter. When developing the program, it was taken into account that user input cannot lead to critical consequences, therefore, to continue working, you only need to call the last command from the console. The user is notified of his actions and sees them on the screen. An example is the appeal of checking the status of a game example file (Figure B.9).

```
Enter 'y' if you made your setup in example.xlsx.
Otherwise press 'n'.
```

Figure B.9 – System asks user to check example file

## **Appendix C**

(recommended)

### **User's manual**

#### **C.1 Introduction**

This software is used in the field of artificial neural networks. The user is given the opportunity to manage instances of the TD-Gammon ANN. Using the software requires an advanced level of training, as it was developed for research. Before using the user must familiarize themselves with this application.

#### **C.2 Purpose and conditions of use**

The program is designed to automate the work with artificial neural networks. The following activities in this area were automated:

- creation of new instances of ANN;
- training instances of ANN;
- the use of these instances for playing backgammon.

The necessary system requirements for working with the software are given in Table C.1. The user is expected to have a higher than average personal computer level.

Table C.1 – Compare instances with most experienced use case description

	Minimum requirements	Recommended Requirements
CPU	1,5 GHz	2,2 GHz
RAM	384 Mb	1 Gb
HDD free space	128 Mb	512 Mb

User Interaction Devices	keyboard	keyboard
--------------------------	----------	----------

### C.3 Preparations

The following program source files are required for operation:

- interface.py;
- Game.py;
- net.py.

A directory called data is also required. This directory is required for storing instances of ANNs. Without it, the program will not work correctly. The program requires a Python interpreter version 3.5 and higher, as well as all related modules. List of third-party modules required to run:

- matplotlib;
- openpyxl;
- numpy.

These packages are installed using the standard Python pip utility. The remaining libraries used in the program are included in the interpreter and do not need to be installed.

The program starts with the interface.py file. It must be started using the Python interpreter, after which a menu for selecting operations will appear in the console window (Figure C.1).

```

Backgammon game using TD-Gammon
Available options:
  1. Make a new instance of ANN
  2. Continue process of machine learning
  3. Get hint for one turn
  4. Compare two ANNs
  5. Compare all ANNs to most experienced
  6. Search for old examples
  7. Configure instance of ANN
Enter option number to continue
Press X to exit
Enter option number:

```

Figure C.1 – The first screen when the program starts successfully

## **C.4 Description of operations**

Almost all software functions require entering the number of games and selecting instances of ANNs. Typical screens are shown in Figures C.2 and C.3. The user is given a choice of 7 operations:

1) Make a new instance of ANN. When this option is selected, a new instance of the ANN is created and a request for entering the number of games for its training will be requested;

2) Continue process of machine learning. This option requires the selection of an INS instance to continue learning and enter the number of games on which it will be trained;

3) Get a hint for one turn. Choosing this option, the user must make sure that the file with an example of the state of the board is ready and make the appropriate input. After that, he will be asked to choose a copy of the ANN that will predict the best move;

4) Compare two ANNs. The user needs to select two instances of the ANN that will play among themselves the entered number of games;

5) Compare all ANNs to most experienced. The user is expected to enter only the number of games. The program itself determines the most trained instance and compares it with all others. The output displays the percentage of wins and the number of completed moves;

6) Search for old examples. This option is designed to read ANN configurations outlined in the old format. You must enter the path to the folder in which the search will occur;

7) Configure instance of ANN. The user must select an instance of the ANN and then indicate the column numbers for truncation. This instance will be saved in a separate file.

```

Point #20 has 0 pieces: []
Point #21 has 0 pieces: []
Point #22 has 0 pieces: []
Point #23 has 6 pieces: ['white', 'white', 'white', 'white', 'white', 'white']
Game #:5
2020-06-08 18:52:57.861219
Black player gets the first turn
Game over in 429 moves
Num states: 214
black won
Point #0 has 0 pieces: []
Point #1 has 0 pieces: []
Point #2 has 0 pieces: []
Point #3 has 0 pieces: []
Point #4 has 0 pieces: []
Point #5 has 0 pieces: []
Point #6 has 0 pieces: []
Point #7 has 0 pieces: []
Point #8 has 0 pieces: []
Point #9 has 0 pieces: []
Point #10 has 0 pieces: []
Point #11 has 0 pieces: []
Point #12 has 0 pieces: []
Point #13 has 0 pieces: []
Point #14 has 0 pieces: []
Point #15 has 0 pieces: []
Point #16 has 0 pieces: []
Point #17 has 0 pieces: []
Point #18 has 0 pieces: []
Point #19 has 0 pieces: []
Point #20 has 0 pieces: []
Point #21 has 0 pieces: []
Point #22 has 0 pieces: []
Point #23 has 11 pieces: ['white', 'white', 'white', 'white', 'white', 'white', 'white', 'white', 'white', 'white', 'white']

```

Figure C.2 – Results of game output

```

1.      1024 - experience
      19:24 - 29/Dec/2019 - creation date
-----
2.       128 - experience
      16:45 - 28/Dec/2019 - creation date
-----
3.       16 - experience
      13:17 - 28/Dec/2019 - creation date
-----
4.        1 - experience
      12:47 - 28/Dec/2019 - creation date
-----
5.      2048 - experience
      02:04 - 31/Dec/2019 - creation date
-----
6.       256 - experience
      20:29 - 28/Dec/2019 - creation date
-----
7.        2 - experience
      12:49 - 28/Dec/2019 - creation date
-----
8.       32 - experience
      13:51 - 28/Dec/2019 - creation date
-----
9.     4096 - experience
      15:31 - 02/Jan/2020 - creation date
-----
10.       4 - experience
      12:53 - 28/Dec/2019 - creation date
-----
11.     512 - experience
      04:09 - 29/Dec/2019 - creation date
-----
12.       64 - experience
      14:47 - 28/Dec/2019 - creation date
-----
13.    8057 - experience
      15:16 - 07/Jan/2020 - creation date
-----
Enter instance number:

```

Figure C.3 – Display of available instances