EDUCATION AND SCIENCE MINISTRY OF UKRAINE
NATIONAL AVIATION UNIVERSITY
DEPARTMENT OF COMPUTER INTEGRATED COMPLEXES

ADMIT TO DEFENSE
Head of department
Viktor M. Sineglazov
“____” ______________ 2020

MASTER’S THESIS

(EXPLANATORY NOTE)

GRADUATE OF EDUCATION AND QUALIFICATION LEVEL “MASTER”

THEME: SUBSYSTEM OF DECISION MAKING ON THE BASIS OF FUZZY NEURAL NETWORK

Executor:	**Koniushenko R.S.**
Supervisor	**Professor Sineglazov V.M.**
Advisor on environmental protection:	**Frolov V.F.**
Advisor on labor protection:	**Konovalova O. V.**
Norms inspector:	**Ph.D., Associate Professor Tupitsyn M.F.**

KYIV 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНО-ІНТЕГРОВАНИХ КОМПЛЕКСІВ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
В.М. Синєглазов
“_____”____________2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-КВАЛІФІКАЦІЙНОГО РІВНЯ
“МАГІСТР”

Тема: ПІДСИСТЕМ ПРИЙНЯТТЯ РІШЕНЬ НА БАЗІ НЕЧІТКИХ НЕЙРОННИХ МЕРЕЖ

Виконавець: **Конюшенко Р.С.**

Керівник: **д.н., професор, Синєглазов В. М.**

Консультант з екологічної безпеки: **Фролов В. Ф.**

Консультант з охорони праці: **Коновалова О. В.**

Нормоконтролер: **к.т.н, доцент, Тупіцин М. Ф.**

Київ 2020

NATIONAL AVIATION UNIVERSITY

Faculty of aeronavigation, electronics and telecommunications

Department of Aviation Computer Integrated Complexes

Educational level master

Field of study: 15 "Automation and Instrumentation"

APPROVED BY

Head of department

Victor M. Sineglazov

"_____"______________ 2020

Graduate Student's Diploma Thesis Assignment

Student's name: Koniushenko Roman

1. The thesis title: "Subsystem of decision making on the basis of fuzzy neural network".

2. The thesis to be completed between: from 01.09.2020 to 18.12.2020

3. Background to the project (work): To focus on the most famous algorithms of neural fuzzy logic.

4. The content of the explanatory note (the list of problems to be considered):

1. Analysis of issues and relevance. 2. Study of the ecological state of the environment. 3. Neural fuzzy network. General terms. 4 Construction principle and complex of technical means of neural fuzzy network; 5. Data flow analysis. 6. Algorithmic application of neural fuzzy network. 7. The structure of the developed system. 8. Development of system software.

5. List of required graphics:

1. Structural diagrams of existing neural networks; 2. Structural diagrams of existing neural fuzzy network; 3. Structural diagrams of developed neural fuzzy network.

6. Planned schedule:

№	**Task**	Deadline	Performance mark
1	Analysis of the urgency of the problem. Formulation of the problem	10.10-20.10.2020	**Done**
2	Study of existing fuzzy classifiers	20.10-01.11.2020	**Done**
3	Algorithmic application of decision-making subsystem based on fuzzy neural networks	01.11-30.11.2020	**Done**
4	Forming conclusions about the work performed	01.12-18.12.2020	**Done**

7. Special chapters' advisors

Chapter	Advisor (position, name)	Date, signature	
		Assignment issue date	Assignment accepted
Labor protection	Konovalova O. V.		
Environmental protection	Frolov V.F.		

8. Date of task receiving: ______________________________________

Diploma thesis supervisor ______________ Viktor M. Sineglazov
(signature)

Issued task accepted ________________ Roman S. Koniushenko
(signature)

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра авіаційних комп’ютерно - інтегрованих комплексів

Освітній ступінь магістр

Напрям 15 – Автоматизація та приладобудування

ЗАТВЕРДЖУЮ

Завідувач кафедри

Синєглазов В.М.

“ ____ ” __________2020 р.

ЗАВДАННЯ
на виконання дипломної роботи студента
Конюшенко Романа Сергійовича

1. Тема роботи: «Підсистема прийняття рішень на базі нечітких нейронних мереж»

2. Термін виконання роботи: з 01.09.2020р. до 18.12.2020.

3. Передумови проекту (роботи): Зосередити увагу на найвідоміших алгоритмах нейронної нечіткої логіки.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1. Аналіз питань та актуальність. 2. Вивчення екологічного стану навколишнього середовища. 3. Нейронна нечітка мережа. Загальні умови. 4 Принцип побудови та комплекс технічних засобів нейронної нечіткої мережі; 5. Аналіз потоку даних. 6. Алгоритмічне застосування нейронної нечіткої мережі. 7. Структура розробленої системи. 8. Розробка системного програмного забезпечення.

5. Перелік обов’язкового графічного матеріалу:
1. Структурні діаграми існуючих нейронних мереж; 2. Структурні діаграми існуючої нейронної нечіткої мережі; 3. Структурні діаграми розвиненої нейронної нечіткої мережі.

6. Календарний план-графік

№	**Задачі**	**Кінцевий термін**	**Стан**
1	Аналіз актуальності проблеми. Постановка задачі	10.10-20.10.2020	**Виконано**
2	Дослідження існуючих нечітких класифікаторів	20.10-01.11.2020	**Виконано**
3	Алгоритмічне застосування підсистеми прийняття рішень на основі нечітких нейронних мереж	01.11-30.11.2020	**Виконано**
4	Формування висновків про виконану роботу	01.12-18.12.2020	**Виконано**

7. Консультанти зі спеціальних розділів

Розділ	Консультант (посада, П. І. Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Охорона праці	Коновалова О.В,		
Охорона навколишнього середовища	Фролов В.Ф.		

8. Дата видачі завдання ______________________________

Керівник: д.т.н., професор ______________________ Синєглазов В.М
(підпис)

Завдання прийняв до виконання ______________________ Конюшенко Р.С.
(підпис)

ABSTRACT

Explanatory note to the thesis "Decision-making subsystem based on fuzzy neural networks" contains pages-118, sections-6, figures-35 , tables-4, used sources-23.

Keywords - neural network; fuzzy network; neural network ensembles; boosting; beging;

The purpose of scientific work: development of a subsystem for decision-making on the basis of fuzzy neural networks, improvement of existing algorithms.

The thesis considers theoretical and software part of the development of the decision-making subsystem for solving the classification problem. The author substantiates the relevance of using fuzzy neural networks to solve the problem of classification, analyzes the existing topologies of fuzzy neural networks and fuzzy classifiers, basic algorithms to improve results and combine them into a single structure, identified their shortcomings and proposed a solution to eliminate them

An optimization and improvement algorithm for solving the classification problem based on the creation of an ensemble of fuzzy neural networks, namely, a fuzzy TSK classifier, is proposed. This software architecture allows you to create a neural classifier that improves the results of an existing solution. And expands the range of calculations performed to classify the input data.

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Підсистема прийняття рішень на базі нечітких нейронних мереж» містить сторінок-118, розділів-6, рисунків-35, таблиць-4, використаних джерел-23.

Ключові слова – нейронна мережа; нечітка мережа; ансамблі нейронних мереж; бустінг; бегінг;

Мета наукової роботи: розробка підсистеми для прийняття рішень на базі нечітких нейронних мереж, покращення існуючих алгоритмів.

В дипломній роботі розглядається теоретична та програмна частина розробки підсистеми прийняття рішень для розв’язання задачі класифікації. Автором обґрунтовано актуальність використання нечітких нейронних мереж для вирішення задачі класифікації, проведено аналіз існуючих топологій нечітких нейронних мереж та нечітких класифікаторів, основних алгоритмів для покращення результатів та поєднання їх в єдину структуру, виявлено їх недоліки та запропоноване рішення, що дозволяє їх усунути

Запропоновано алгоритм оптимізації та покращення для вирішення задачі класифікації на основі створення ансамблю з нечітких нейронних мереж а саме, нечіткого класифікатора TSK. Дана програмна архітектура дозволяє створити нейронний класифікатор який покращує результати уже існуючого рішення. Та розширює спектр виконуваних обчислювань для класифікації вхідних даних.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНО-ІНТЕГРОВАНИХ КОМПЛЕКСІВ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
В.М. Синєглазов
"_____"____________2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-КВАЛІФІКАЦІЙНОГО РІВНЯ "МАГІСТР"

Тема: ПІДСИСТЕМ ПРИЙНЯТТЯ РІШЕНЬ НА БАЗІ НЕЧІТКИХ НЕЙРОННИХ МЕРЕЖ

Виконавець: **Конюшенко Р.С.**

Керівник: **д.н., професор, Синєглазов В. М.**

Консультант з екологічної безпеки: **Фролов В. Ф.**

Консультант з охорони праці: **Коновалова О. В.**

Нормоконтролер: **к.т.н, доцент, Тупіцин М. Ф.**

Київ 2020

NATIONAL AVIATION UNIVERSITY

Faculty of aeronavigation, electronics and telecommunications

Department of Aviation Computer Integrated Complexes

Educational level master

Field of study: 15 “Automation and Instrumentation"

APPROVED BY

Head of department

Victor M. Sineglazov

“_____”______________ 2020

Graduate Student’s Diploma Thesis Assignment

Student’s name: Koniushenko Roman

1. The thesis title: “Subsystem of decision making on the basis of fuzzy neural network”.

2. The thesis to be completed between: from 01.09.2020 to 18.12.2020

3. Background to the project (work): To focus on the most famous algorithms of neural fuzzy logic.

4. The content of the explanatory note (the list of problems to be considered):

1. Analysis of issues and relevance. 2. Study of the ecological state of the environment. 3. Neural fuzzy network. General terms. 4 Construction principle and complex of technical means of neural fuzzy network; 5. Data flow analysis. 6. Algorithmic application of neural fuzzy network. 7. The structure of the developed system. 8. Development of system software.

5. List of required graphics:

1. Structural diagrams of existing neural networks; 2. Structural diagrams of existing neural fuzzy network; 3. Structural diagrams of developed neural fuzzy network.

6. Planned schedule:

№	**Task**	Deadline	Performance mark
1	Analysis of the urgency of the problem. Formulation of the problem	10.10-20.10.2020	**Done**
2	Study of existing fuzzy classifiers	20.10-01.11.2020	**Done**
3	Algorithmic application of decision-making subsystem based on fuzzy neural networks	01.11-30.11.2020	**Done**
4	Forming conclusions about the work performed	01.12-18.12.2020	**Done**

7. Special chapters' advisors

Chapter	Advisor (position, name)	Date, signature	
		Assignment issue date	Assignment accepted
Labor protection	Konovalova O. V.		
Environmental protection	Frolov V.F.		

8. Date of task receiving: ______________________________________

Diploma thesis supervisor ______________ Viktor M. Sineglazov
(signature)

Issued task accepted ________________ Roman S. Koniushenko
(signature)

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра авіаційних комп’ютерно - інтегрованих комплексів

Освітній ступінь магістр

Напрям 15 – Автоматизація та приладобудування

ЗАТВЕРДЖУЮ

Завідувач кафедри

Синєглазов В.М.

“ ____ ” __________2020 р.

ЗАВДАННЯ
на виконання дипломної роботи студента
Конюшенко Романа Сергійовича

1. Тема роботи: «Підсистема прийняття рішень на базі нечітких нейронних мереж»

2. Термін виконання роботи: з 01.09.2020р. до 18.12.2020.

3. Передумови проекту (роботи): Зосередити увагу на найвідоміших алгоритмах нейронної нечіткої логіки.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1. Аналіз питань та актуальність. 2. Вивчення екологічного стану навколишнього середовища. 3. Нейронна нечітка мережа. Загальні умови. 4 Принцип побудови та комплекс технічних засобів нейронної нечіткої мережі; 5. Аналіз потоку даних. 6. Алгоритмічне застосування нейронної нечіткої мережі. 7. Структура розробленої системи. 8. Розробка системного програмного забезпечення.

5. Перелік обов’язкового графічного матеріалу:

1. Структурні діаграми існуючих нейронних мереж; 2. Структурні діаграми існуючої нейронної нечіткої мережі; 3. Структурні діаграми розвиненої нейронної нечіткої мережі.

6. Календарний план-графік

№	**Задачі**	**Кінцевий термін**	**Стан**
1	Аналіз актуальності проблеми. Постановка задачі	10.10-20.10.2020	**Виконано**
2	Дослідження існуючих нечітких класифікаторів	20.10-01.11.2020	**Виконано**
3	Алгоритмічне застосування підсистеми прийняття рішень на основі нечітких нейронних мереж	01.11-30.11.2020	**Виконано**
4	Формування висновків про виконану роботу	01.12-18.12.2020	**Виконано**

7. Консультанти зі спеціальних розділів

Розділ	Консультант (посада, П. І. Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Охорона праці	Коновалова О.В,		
Охорона навколишнього середовища	Фролов В.Ф.		

8. Дата видачі завдання ______________________________

Керівник: д.т.н., професор ______________________________ Синєглазов В.М
(підпис)

Завдання прийняв до виконання ______________________________ Конюшенко Р.С.
(підпис)

ABSTRACT

Explanatory note to the thesis "Decision-making subsystem based on fuzzy neural networks" contains pages-118, sections-6, figures-35 , tables-4, used sources-23.

Keywords - neural network; fuzzy network; neural network ensembles; boosting; beging;

The purpose of scientific work: development of a subsystem for decision-making on the basis of fuzzy neural networks, improvement of existing algorithms.

The thesis considers theoretical and software part of the development of the decision-making subsystem for solving the classification problem. The author substantiates the relevance of using fuzzy neural networks to solve the problem of classification, analyzes the existing topologies of fuzzy neural networks and fuzzy classifiers, basic algorithms to improve results and combine them into a single structure, identified their shortcomings and proposed a solution to eliminate them

An optimization and improvement algorithm for solving the classification problem based on the creation of an ensemble of fuzzy neural networks, namely, a fuzzy TSK classifier, is proposed. This software architecture allows you to create a neural classifier that improves the results of an existing solution. And expands the range of calculations performed to classify the input data.

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Підсистема прийняття рішень на базі нечітких нейронних мереж» містить сторінок-118, розділів-6, рисунків-35, таблиць-4, використаних джерел-23.

Ключові слова – нейронна мережа; нечітка мережа; ансамблі нейронних мереж; бустінг; бегінг;

Мета наукової роботи: розробка підсистеми для прийняття рішень на базі нечітких нейронних мереж, покращення існуючих алгоритмів.

В дипломній роботі розглядається теоретична та програмна частина розробки підсистеми прийняття рішень для розв’язання задачі класифікації. Автором обґрунтовано актуальність використання нечітких нейронних мереж для вирішення задачі класифікації, проведено аналіз існуючих топологій нечітких нейронних мереж та нечітких класифікаторів, основних алгоритмів для покращення результатів та поєднання їх в єдину структуру, виявлено їх недоліки та запропоноване рішення, що дозволяє їх усунути

Запропоновано алгоритм оптимізації та покращення для вирішення задачі класифікації на основі створення ансамблю з нечітких нейронних мереж а саме, нечіткого класифікатора TSK. Дана програмна архітектура дозволяє створити нейронний класифікатор який покращує результати уже існуючого рішення. Та розширює спектр виконуваних обчислювань для класифікації вхідних даних.

CONTENT

Glossary

NN- neural network;

ANN – artificial neural network;

FNN – fuzzy neural network;

TSK – Takagi Sugeno Kang network;

INTRODUCTION

Research on artificial neural networks (hereinafter - neural networks) is due to the fact that the method of processing information by the human brain is significantly different from the methods used by conventional digital computers. The brain is an extremely complex, nonlinear, parallel computer (information processing system). It has the ability to organize its structural components, called neurons, so that they can perform specific tasks (such as pattern recognition, signal processing, motor functions) several times faster than the fastest modern computers can afford. . An example of such a task of information processing can be ordinary vision. The function of the visual system is to create an idea of the world around us in a form that provides the opportunity to interact with this world. More precisely, the brain consistently performs a number of recognition tasks (for example, recognizing a familiar person in an unfamiliar environment). It takes him about 100-200 milliseconds, while performing similar tasks of even less complexity on a computer can take several days.

The concept of neuronal development is associated with the concept of brain plasticity - the ability to adjust the nervous system in accordance with environmental conditions. It is plasticity that plays the most important role in the work of neurons as units of information processing in the human brain. Similarly, in artificial neural networks, work is done with artificial neurons. In the general case, the neural network is a machine that simulates the way the brain processes a specific task. This network is usually implemented using electronic components or simulated by a program running on a digital computer.

It is obvious that neural networks derive their strength, firstly, from the parallelization of information processing and, secondly, from the ability to self-learn, i.e. to create generalizations. The term generalization refers to the ability to obtain a reasonable result on the basis of data that do not occur in the learning process. These properties allow neural networks to solve complex (large-scale) problems that are currently considered difficult to solve. However, in practice, neural networks cannot provide ready-made solutions during autonomous operation. They need to be integrated into complex systems.

Fuzzy set methods are especially useful in the absence of an accurate mathematical model system operation. The theory of fuzzy sets makes it possible to apply to acceptance decisions inaccurate and subjective expertise on the subject area without formalizing them in the form of traditional mathematical models.

Using the theory of fuzzy sets, the issues of reconciling contradictions are solved decision-making criteria, creation of logical regulators of systems. Fuzzy sets make it possible to apply a linguistic description of complex processes, establish fuzzy relationships between concepts, predict the behavior of the system, form a set of alternative actions, perform formal description of fuzzy decision-making rules.

Fuzzy set theory methods are a convenient tool for designing interfaces in human-machine systems. On the basis of fuzzy logical derivation control systems are built, presentation of knowledge, decision support, approximation, structural and parametric identification, pattern recognition, optimization. Fuzzy logic finds application in consumer electronics, diagnostics, various expert systems. Fuzzy expert systems to support decision-making are widely used in military affairs, medicine and economy. They are used to carry out business forecasting, risk assessment and profitability investment projects. Based on fuzzy logic, global policy decisions and simulate crisis situations[7-8].

The prospect of using fuzzy logic is to develop hybrid artificial methods intelligence, which include fuzzy artificial neural networks, adaptive replenishment of fuzzy databases rules, support of fuzzy queries to databases, construction of fuzzy cognitive maps, fuzzy graphs, fuzzy Petri nets, fuzzy decision trees, fuzzy clustering, etc. [13 - 15].

Relevance of theme - modern technologies, such as artificial neural networks, allow humanity to be more productive in all areas of life. Neural networks are used to optimize work in medicine, trade, banking and more. In general, neural networks can solve the problems of pattern recognition and solving the problem of classification. Global giants such as Google, IBM, Apple are creating their own algorithms and networks to improve existing processes. It is obvious that Ukrainian science must keep up with world trends, because the use of such systems can be useful not only for commercial purposes, but also for government agencies. This qualification work is aimed at the development and identification of prospects for the use of such classifiers.

The purpose and objectives of the thesis – to develop a subsystem for decision-making based on fuzzy neural networks to solve the classification problem.

The object of study of this work is a structural-parametric synthesis of classifiers based on fuzzy neural networks.

The subject of research there is a subsystem for decision-making, based on fuzzy neural networks.

CHAPTER 1
THE USE OF ARTIFICAIL INTELLIGENCE IN DECISION-MAKING TASKS

Artificial neural networks - is a mathematical model of the functioning of neural networks traditional for living organisms, which are networks of nerve cells. As in the biological analogue, in artificial networks, the main element is neurons, interconnected and forming layers, the number of which can be different depending on the complexity of the neural network and its purpose (tasks to be solved).

Perhaps the most popular task of neural networks is visual pattern recognition. Today networks are being created in which machines are able to successfully recognize symbols on paper and bank cards, signatures on official documents, detect objects, etc. These functions make it possible to significantly facilitate human labor, as well as increase the reliability and accuracy of various work processes due to the absence of the possibility of making mistakes due to the human factor.

1.1. Solving the problem of decision making as a means of improving efficiency

The classification is the task of splitting a set of objects or observations into a priori given groups, called classes, inside each of which they are considered to be similar to each other, and have approximately the same properties and attributes. A finite set of objects is specified, for which it is known which classes they belong to. This set is called sampling. To what class other objects belong to unknowns. In this case, the solution is based on the analysis of the values of attributes (attributes), on which basis it is under construction.

An algorithm that will be able to classify an arbitrary object from the source set. Classification is one of the most important tasks in Data Mining.

Data Mining is a multidisciplinary field that originates and develops on the basis of such sciences as applied statistics, image recognition, artificial intelligence, database theory, and so on. Data Mining Technology is a process of detecting in the "raw" data previously unknown, non-trivial, practically useful and accessible interpretations of knowledge necessary for decision-making in various spheres of human activity.

Data Mining technology is designed to search for large volumes of data that are non-obvious, objective and useful in practice. Among the numerous applications of the classification problem, one can distinguish the following: classification of images, classification of data, recognition, handwriting, speech recognition, medical diagnosis, bankruptcy prediction, troubleshooting, evaluation creditworthiness of borrowers, and many other areas.

If the number of classes is limited to two, then there is a binary classification, which can be reduced to much more complex tasks. For example, instead of defining such credit levels risk, such as "High", "Medium" or "Low", you can use only two - "Issue" or "Refuse". For classification using mathematical methods it is necessary to have a formal description of the object that can be operated, using the mathematical apparatus of classification. With such a description the database most often acts. Each database entry carries information about some property of the object. The set of input data (data sample) is divided into two sets: training (training set) and test (test set). The training sample includes objects for which the values are known as independent and dependent variables. Based on the training sample a model for determining the value of the dependent variable is built. It's often called the classification function. To get the most accurate functions to the training sample are subject to the following basic requirements:

- the number of objects included in the sample should be sufficiently great. The more objects, the built on their basis the classification function will be more accurate;
- the sample must include objects that represent all possible classes;
- For each class the sample must have a sufficient number of objects.

The test set also contains input and output values parameters. Here the original values are used for verification workability of the model.

The classification process consists of two stages: design model and its use.

- The design of the model is based on training samples. As a result, we obtain the model that is presented or classification rules or decision tree or mathematical formula or computer object (as neural networks).
- Use of the model: classification of new or unknown values.
- Assessment of the correctness (accuracy) of the model. Famous the values from the test set are compared with the results use of the received model. For the level of accuracy the percentage of correctly classified examples in the test set.
- If the accuracy of the model is acceptable, it is possible to use the model to classify new examples, class which is unknown.

The main problems encountered in solving problems classification - is the unsatisfactory quality of the input data in which there are both erroneous data and omitted values, different types attributes - numerical and categorical, different significance of attributes, and also the so-called overfitting and underfitting problems.

The essence of the first of them is that the classification function when constructing "adapts well" to the data, and the errors that occur in them, and tries to interpret anomalous values as part internal data structure. Obviously, such a model will incorrectly work further with other data where the nature of the errors will be somewhat different. This situation is manifested in the fact that the error on the test set has much more error on the training. The term underfitting indicates a situation where there is too much error when checking the classifier on the training plural. This means that there are no special patterns in the data detected and either do not exist at all, or you must choose another method of them detection.

Unclear - this means that the patterns found are not the standard methods of processing information or expert way.

Objective - this means that the revealed patterns will fully correspond to reality, in contrast to the expert opinion, which is always subjective.

Practically useful - this means that the conclusions have a specific meaning, which can be found in practical application.

The basis of Data Mining technology is the concept of patterns (patterns), which are the regularities inherent in subsets of data, which can be expressed in a form that is understandable to a person.

In each subject area, upon closer examination, you can find the formulations of tasks for neural networks. Here is a list of selected areas where solving this kind of problem is of practical importance right now.

- Economy and business: forecasting time series (exchange rates, commodity prices, demand, sales volumes, ...), automatic trading (trading on the foreign exchange, stock or commodity exchange), assessing the risks of non-repayment of loans, predicting bankruptcies, assessing the value of real estate, identifying overvalued and undervalued companies, rating, optimization of commodity and cash flows, reading and recognizing checks and documents, security of transactions with plastic cards.
- Medicine and healthcare: making a diagnosis to a patient (diagnosing diseases), processing medical images, cleaning instrument readings from noise, monitoring the patient's condition, predicting the results of using different methods of treatment, analyzing the effectiveness of treatment.
- Avionics: trainable autopilots, radar signal recognition, adaptive piloting of a heavily damaged aircraft, unmanned aerial vehicles (drones).
- Communication: video compression, fast encoding and decoding, optimization of cellular networks and packet routing schemes.
- Internet: associative search for information, electronic secretaries and autonomous agents on the Internet, filtering and blocking spam, automatic rubricating of messages from news feeds, targeted advertising and marketing for e-commerce, captcha recognition.
- Production automation: optimization of production process modes, product quality control, monitoring and visualization of multidimensional dispatch information, prevention of emergency situations.

- Robotics: recognizing the scene, objects and obstacles in front of the robot, laying the route of movement, controlling the manipulators, maintaining balance.
- Political and sociological research: predicting election results, analyzing polls, predicting the dynamics of ratings, identifying significant factors, clustering the electorate, studying and visualizing the social dynamics of the population.
- Security, security systems: face recognition; identification of a person by fingerprint, voice, signature or face; license plate recognition; monitoring information packets and information flows in a computer network to detect intrusions; detection of counterfeits; analysis of data from video cameras and various sensors; analysis of aerospace imagery (for example, to detect forest fires or illegal logging).
- Input and processing of information: recognition of handwritten texts, scanned postage, payment, financial and accounting documents; recognition of speech commands, speech input of text into a computer.
- Geological exploration: analysis of seismic data, associative methods of prospecting for minerals, estimation of field resources.
- Computer and board games: the creation of neuro-players in checkers and chess (ratings confirmed by playing with people - at the level of masters and international masters), winning in Go from champions of Europe and the world, on average better than a person, passing almost fifty old classic games with Atari (all sorts of Pongs, Packman's, ...).

The abundance of the above areas of application of neural networks is not a publicity stunt. It's just that neural networks are a flexible and powerful set of tools for solving various tasks of data processing and analysis.

1.2. Statement of the classification problem

As a science, the theory of pattern recognition began to emerge somewhere in the late 50's. At first it was a meaningful statement of the problem. It was necessary to build a car that could classify different situations as it was done by living beings. Such a general statement of the problem led to the fact that there were different directions of research. Some scientists have built a model for the process of perception. Others thought that the main thing - is the creation of algorithms learning to recognize images to solve specific problems. Some were looking for new mathematical problems. If at first quite successfully managed to advance in all directions, then over time, successes significantly decreased.

There are two opposing points of view on the problem of recognition. The first was that it was necessary to find such a description of objects, using a priori data about them, that finding the classification principle would no longer be difficult.

The second saw the main problem in finding the classification rule among a given set of deciding rules. So, the question arose: are the only principles of constructing an adequate description of images of different nature? If so, the pattern recognition theory is the discovery of these principles. If not, then pattern recognition theory is the task of minimizing average risk in a special class of decisive rules. Most scholars hold the second point of view, which is to minimize the risk in a special class of decisive rules.

At present, NNs are used to solve problems of various types, for example, tasks of approximation, classification, image recognition, forecasting. Along with the classical approach, there are a number of methods for solving these problems, which simplifies or accelerates the calculation. One such method is the transition to fuzzy logic.

Let X – set of object descriptions, Y – a set of class names. There is a target addiction - reflection $y^*: X \to Y,$ the meaning of which is known on the objects of the training sample $X^m = \{(x_1, y_1), \dots, (x_m, y_m)\}$. Need to build algorithm (neural network) $a: X \to Y$, capable of classifying an arbitrary object $x \in X$ [4].

Setting the task of classification

The main approaches to solving the classification problem are the following: decision trees, reference vectors, closest neighbor, CBR-method, linear regression, neural network method [1]. Analysis of these methods showed
that the most effective is the neural network method.

As a neural network, for constructing a classifier it is expedient to use hybrid type of network: NEFCLASS, NEFCON, NEFPROX, TSK, whose work is based on fuzzy logic.

Further improvement of the efficiency of such classifiers can be achieved through in-depth training (see Section 3.1).

In [3], J. Hinton proved that the study of deep neural networks is possible with the help of "greedy" layer training. The essence of "greedy" learning is finding locally-optimal solutions for each sub task, expecting the end result to be optimal in the end. Sub-task in the process of training the neural network is to study a single layer.

In [2] J. Hinton proposed the concept of training each layer separately with the help of an auto associated with the subsequent study of the network by a standard method, for example, the method of reverse error propagation.

Automobile associate networks are networks that, in the output, should be as accurate as possible representing the data that is fed to the network. For problems of deep learning, limited machines of Boltzmann and auto converters are used. So, summing up the information, we modem out the following statements:

Classify an object - then, specify the number (or class name) to which this object belongs.

Classification of an object - the number or class name, issued by the classification algorithm as a result of its application to this particular object.

Methods for solving the problem of classification are conventionally divided into two groups.

Statistical methods of classification:

- Bayesian (naive) classification;
- logistic regression;
- discriminatory analysis

Methods of machine learning

- classification using decision trees;
- classification using artificial neural networks;
- classification using coating algorithms;
- classification by the method of reference vectors;
- classification using the method of k-nearest neighbors;

- classification by CBR-method.

1.3. Review of methods

Each of the above approaches has both advantages and limitations. In particular, in the traditional statistical procedure of classification based on the Bayesian decision-making theory, the main limitation is the dependence of the method on a large number of conditions and conditions under which this model operates.

Concerning decision-making trees, especially with a large number of branches, they are characterized by complexity for realization and understanding, time spent in the training phase.

Classification by the nearest neighbor method is often inaccurate in the event of a redundancy or inconsistency of characteristic features.

The genetic algorithm is not effective to find the optimal value, designed to find the seamless result. Among the above methods, one can distinguish the neural network approach to the solution of the classification problem, which is considered in many scientific studies. Many classifiers of neural network architecture have been created, which are now widely used to solve medical diagnosis, image classification, data classification, bankruptcy prediction, and so on. These technologies become indispensable in the business, industry and science fields. Let's consider more basic methods for solving the problem of classification.

Subject-oriented analytical systems: Subject-oriented analytical systems are very diverse. The broadest subclass of such systems that has become widespread in the field of financial market research is called "technical analysis". It represents a set of dozens of methods for predicting price dynamics and choosing the optimal structure of an investment portfolio based on various empirical models of market dynamics. These methods often use a simple statistical apparatus, but maximally take into account the specifics of their area (professional language, systems of various indexes, etc.). There are many programs in this class on the market. As a rule, they are quite cheap. 53 Statistical Packages The latest versions of almost all known statistical packages include, along with traditional statistical methods, IAD elements. But their focus is still on the classical methods - correlation, regression, factor analysis, and so on. The disadvantage of the systems of this class is the requirement for special training of the user. It is also noted that powerful modern statistical packages are too "heavyweight" for mass applications in finance and business. In addition, these systems are often expensive. There is even a more serious lack of statistical packets, limiting their use in IAD. Most of the methods in the package are based on a statistical paradigm in which the averages of the sample serve as the main contributors. And these characteristics, as indicated above, in the study of real complex life phenomena are often fictitious quantities. Examples of the most powerful and most commonly used statistical packages are SAS (SAS Institute), SPSS (SPSS), STATGRAPICS (Manugistics), STATISTICA, STADIA, and others.

Neural Networks: This is a large class of systems whose architecture has an analogy (as is now known, rather weak) with the construction of neuronal tissue from neurons. In one of the most common architectures, the multilayer perceptron with the inverse spread of error, the work of neurons in the hierarchical network is imitated, where each higher level neuron is connected by

its inputs to the outputs of the lower layer neurons. On the neurons of the lowest layer are input values of input parameters, on the basis of which it is necessary to make some decisions, to predict the development of the situation, etc. These values are considered as signals transmitted to the next layer, loosening or amplifying, depending on the numerical values (weights) attributed to the intraneuronal bonds. As a result, at the output of the neuron of the highest upper layer, a certain value is considered, which is considered as the response - the response of the entire network to the input values of the input parameters. In order for the network to continue to be used, it must be "trained" on previously received data, for which the input parameters are known, and the correct answers to them. The training is to select the weights of intraneuronal relationships, the answers that provide the closest network to the known correct answers. The main disadvantage of the neural network paradigm is the need for a very large amount of training sample. Another significant drawback is that even a trained neural network is a "black box". Knowledge that is recorded as the weight of several hundred intraneuronal relationships is not at all a subject of analysis and interpretation by a person (known attempts to interpret the structure of the configured neural network look unconvincing - the KINOsuite-PR system). Examples of neural network systems are BrainMaker (CSS), NeuroShell (Ward Systems Group), OWL (HyperLogic).

Systems of reasoning based on similar cases: The idea of reasoning systems based on similar cases (case basis based, CBR) at first glance is extremely simple. In order to make a prediction for the future or to choose the right solution, these systems find in the past similar analogues of the current situation and choose the same answer that was right for them. Therefore, this method is also called the "nearest neighbor" method. Recently, the term memory-based reasoning, which emphasizes the fact that the decision is made on the basis of all information accumulated in memory, has also been disseminated.

CBR systems show good results in a variety of tasks. Their main disadvantage is that they generally do not create any models or rules that summarize previous experience - in choosing a solution they are based on an array of available historical data, therefore it is impossible to say on the basis of which particular factors of the CBR systems build their answers. Another disadvantage is the tyranny allowed by the CBR system when choosing "closeness". From this measure, the most crucial way depends on the amount of precedents that need to be stored in memory to achieve satisfactory classification or prediction. Examples of systems using CBR are KATE tools (Acknosoft, France), Pattern Recognition Workbench (Unica, USA). Decision trees

Trees solution: is one of the most popular approaches to solving tasks of classification. They create a hierarchical structure of rules such as "IF ... TO ..." (if-then), which has the form of a tree. To decide on which class to classify an object or situation, you need to answer the questions in the nodes of this tree, starting with its root. The questions have the form "value of parameter A greater than x?" If the answer is positive, then the transition to the right node of the next level is carried out, if negative - then to the left node, then again it is necessary to put a question related to the corresponding node. but the decision trees are fundamentally unable to find the "best" (the most complete and exactest) rules in the data, they implement the naive principle of consistently revising the signs and "actually" affect "the fragments of the true patterns, creating only The most well-known are See5 / 5.0 (RuleQuest, Australia), Clementine (Integral Solutions, UK), SIPINA (University of Lyon, France), IDIS (Information Discovery, USA).), KnowledgeSeeker (ANGOSS, Canada). The cost of these systems varies from cheap to very expensive.

Evolutionary programming: We illustrate the current state of this approach on the example of PolyAnalyst - the Russian development system, which has received today a general recognition in the market.

In this system, the hypothesis about the type of dependence of the target variable from other variables is formulated in the form of programs in some internal programming language. The process of building programs is built as evolution in the world of programs (this approach is very similar to genetic algorithms). When the system finds a program that more or less satisfactorily expresses the desired dependence, it starts to make small modifications to it and selects among the built-in affiliate programs those that increase its accuracy. In this way, the system "grows" several genetic lines of programs that compete with one another in the exact expression of the desired dependence. A special module of the PolyAnalyst system translates the found dependencies from the internal language of the system into a user-understandable language (mathematical formulas, tables, etc.). Another direction of evolutionary programming is the search for the dependence of target variables from others in the form of functions of a certain type. For example, in one of the most successful algorithms of this type - the method of group account of arguments (MSUA) - the dependence is sought in the form of polynomials. At this time, from the systems sold in Russia, MSUA is implemented in the NeuroShell system of Ward Systems Group. The cost of systems varies to average price levels.

Genetic Algorithms:. They need to be considered rather as a powerful means of solving various combinatorial problems and optimization tasks. However, genetic algorithms are now included in the standard IAD methodology toolkit, so they are included for consideration. The first step in constructing genetic algorithms is to encode the output

logic patterns in a database called chromosomes, and the whole set of such patterns is called the population of the chromosomes. Next, to implement the concept of selection, a method for mapping
different chromosomes are introduced. The population is processed using reproduction procedures, variability (mutations), genetic composition. These procedures simulate biological processes.

The most important among them are random data mutations in individual chromosomes, transitions (cross-over), and the recombination of genetic material contained in individual parent chromosomes, and the migration of genes. In the course of the work of the procedures at each stage of evolution are the populations with all the better individuals. Genetic algorithms are convenient because they are easy to parallelize. For example, you can break the generations into several groups and work with each of them independently, exchanging from time to time several chromosomes. There are also other methods for parallelizing genetic algorithms. Genetic algorithms have many drawbacks. The criterion for selecting chromosomes and the procedures used is heuristic and far from guaranteeing finding a "better" solution. As in real life, evolution can "jam" on any unproductive industry. And, on the contrary, we can give examples of how two hopeless parents, who will be excluded from the evolution of the genetic algorithm, are capable of making a highly effective descendant. This is especially noticeable when solving high-dimensional problems with complex internal connections. An example is the GeneHunter system from the Ward Systems Group. Its value is credited to the average price category.

The algorithms of limited browsing: The algorithms of the limited search of the past are proposed in the mid 60-ies of the twentieth century. MM Bongard to look for the logical patterns in the data. Since then, they have demonstrated their effectiveness in solving a variety of tasks from all sorts of areas. These algorithms compute the frequencies of combinations of simple logical events in subgroups of data. Examples of simple logical events: X = a; X <a; X a; a <X <b, etc., where X is a parameter, "a" and "b" are constants. The limitation is the length of a combination of simple logical events.

Based on the analysis of computed frequencies, a conclusion is made on the usefulness of a combination for establishing an association in the data, for classification, forecasting, etc.

The most vivid modern representative of this approach is the WizWhy WizSoft enterprise system. Although the author of the Abraham Maydan system does not disclose the specifics of the algorithm underlying the work of WizWhy, based on the results of careful testing of the system, conclusions were drawn about the existence of a limited search (the results were studied, the dependence of the time of their receipt on the number of parameters analyzed, etc.). The author of WizWhy claims that his system detects all logical if-then rules in the data. In fact, this is not true. First, the maximum length of the combination in the if-then-rule in the WizWhy system is 6, and secondly, from the very beginning of the algorithm, a heuristic search for simple logical events is made, on which then all further analysis is under construction. Understanding these features of WizWhy, it was not difficult to offer the simplest test task that the system could not solve at all. Another point - the system issues a solution at an acceptable time, only for a relatively small dimension of the data. However, WizWhy is today one of the leaders in the market for IAD products. This is not without reason. The system continuously demonstrates higher performance when solving practical tasks than all other algorithms. The system cost is slightly higher than average. Systems for visualizing multidimensional data One or another measure of the graphic data image is supported by all SFL systems. In addition, a fairly significant market share is occupied by systems specializing exclusively in this function. An example here is the DataMiner 3D program of the Slovak firm Dimension 5 (5th dimension). In such systems, the main focus is on the user interface tolerance, which allows to associate with the analyzed parameters various parameters of the scatter diagram of the database objects. These parameters include color, shape, orientation relative to their own axis, dimensions and other properties of graphic elements of the image. In addition, data visualization systems are indicated by convenient means for scaling and rotation of images. The cost of visualization systems can be several hundred dollars.

Among the systems described above, neural networks are the most accessible and most accurate means for solving the classification problem, the variety of algorithms for solving is increasing every year, and the networks themselves are becoming more and more popular.

1.4. Formation of a training sample to solve the classification problem

An integral part of modern information systems are databases (DB) designed to store and retrieve information. As a rule, the database, being at the lower level in the structure of a multi-level information system, is a data source for information processing and decision-making facilities. Accordingly, among the requirements for data stored in a database, such as completeness, relevance and reliability come to the fore. Existing approaches to ensuring validity include integrity constraints, triggers, and the use of stored procedures [1, 2]. They involve checking the values stored in the cells for compliance with known, predetermined limits, however, even values within acceptable limits may not reflect the actual characteristics of the described object or phenomenon, thus leading to errors that are expressed in data unreliability.

A more accurate assessment of the reliability of data associated with errors in database tables can be carried out using data mining methods based on the use of artificial neural networks (ANN) [3], the key feature of which is the ability to learn and generalize. A particularly important stage of the method for finding errors in the database is the stage of forming a training sample (or a reference fragment of the database). At the same time, the need to ensure the representativeness of the training sample comes to the fore.

Representativeness is the correspondence of the characteristics of the sample to the characteristics of the population or the general population as a whole [4, 5]. Representativeness determines how much it is possible to generalize the results of a study using a certain sample to the entire general population from which it was collected.

In the context of analytical technologies, the representativeness of the initial data should be understood as the presence of a sufficient number of various training examples reflecting the rules and patterns that should be detected by the model in the learning process [6]. It has three aspects [4]:

- sufficiency - the number of training examples should be sufficient for training. For a neural network, it is necessary that the number of training examples be several times greater than the number of weights of interneural connections, otherwise the model may not acquire the ability to generalize. In addition, the sample size must be sufficient to form the training and test sets;
- variety - a large number of different combinations of input-output in training examples. The ANN's ability to generalize will not be achieved if the number of examples is sufficient, but they are all the same, i.e. representing only a part of the classes characteristic of the original set;
- uniformity of presentation of classes - examples of different classes should be presented in the training sample in approximately the same proportions. If one of the classes prevails, this can lead to "skew" in the learning process of the model, and this class will be determined by the model as the most likely for any new observations [8].

Let's consider each of the requirements in more detail.

Based on the properties of neural networks, the number of neurons in the input layer when using the database table (ratio) for training the ANN is equal to the number of columns in the table selected for testing [6, 8]. It is stored in the database metadata and can be retrieved from service tables [7, 8]. For a multilayer network, the number of neurons in the hidden layer should exceed the number of neurons in the input layer by 1.5 - 2 times [9], thus, the total number of neurons in the input and hidden layers is from 2.5n to 3n. Since the network is fully connected, i.e. each neuron of the previous layer is connected to all neurons of the next layer, then the number of connections between the 1st and 2nd layers is equal to $n \times 2 \times n = 2 \times n^2$.

The next two requirements - diversity and uniformity - can be provided by random selection of rows from the table for the training sample [11].

However, the random selection method cannot guarantee 100% fulfillment of these conditions. The present element of randomness, especially with a sufficiently large number of classes, can introduce significant errors, including those leading to the non-representativeness of the training sample. To some extent, this drawback can be eliminated by forming 2 training samples. If, as a result of clustering two samples, the same number of clusters is obtained, we can talk about their sufficient representativeness.

1.4.1. Ways to improve the sample

Transfer learning is the application of knowledge extracted by a neural network when solving another problem to the solution of a problem.

Deep neural networks require large amounts of data for training convergence. Therefore, there is often a situation when there is not enough data for the problem being solved in order to well train all layers of the neural network. Transfer learning is used to solve this problem [22].

Most often, transfer learning looks like this: several hidden layers are added to the neural network trained for a specific task, which allow using the already acquired knowledge to solve a more specific task. For example, the knowledge gained while learning to recognize various objects can be used to solve the problem of food recognition.

How can Transfer learning help? If you carefully study the process of deep learning, you will understand the following: on the first layers of the network, they try to catch the simplest patterns, on the middle layers - the most important, and on the last layers - specific details. Such trained networks are generally useful in other computer vision problems. Let's see how to implement TL using Keras for various tasks.

Simple implementation with Keras, keeping in mind that the properties of ConvNet are more primitive in the first layers and get more complicated with each subsequent layer, consider four typical scenarios.

- The new dataset is small and similar to the original dataset.

If we try to train the entire network at once, the problem of model overload arises. Since the data is the same as the original, we expect that the higher-level properties in ConvNet will also be relevant to this dataset. Hence, the best strategy might be to train a linear classifier using CNN codes.Thus, let's drop all VGG19 layers and train only the classifier:

```
for layer in model.layers:
   layer.trainable = False
```

- The new dataset is large and similar to the original.

Since we have more data, there is a lot of confidence that we will not overload the model if we try to fine-tune the entire network at once:

```
for layer in model.layers:
   layer.trainable = True
```

If you want to discard the first few layers that form the most primitive properties, you can do so with the following code:

```
for layer in model.layers[:5]:
   layer.trainable = False.
```

- The new dataset is small and very different from the original.

Since the dataset is small, you can export properties from earlier layers and train the classifier on top of them. This requires some knowledge of h5py:

This code should help. It will fetch the "block2_pool" properties. This is usually not very useful since the layer has 64x64x128 properties and training a classifier on top of them may not help. You can add multiple fully connected layers and train the neural network on top of them. This must be done in the following order:

  - Add some FC layers and an output layer.
  - Set the weights for the earlier layers and discard them.
  - Train the network.

- The new dataset is large and very different from the original

Since the dataset is large, you can create your own network or use existing ones. In this case, proceed as follows. Train the network using random initializations, or use pre-trained weights to get started. The second option is usually preferred.

Conclusions: solving of classification tasks is very importing process, lets summarized some key points. Classification is often the foundation on which algorithms for solving more complex problems are built. For example, the classification is used when creating recommendation systems and in particular when implementing collaborative filtering. Safari Reader Mode is another example where classification algorithms are used to achieve the ultimate goal. The essence of this browser mode is that it allows you to automatically remove from the page all the husk that has nothing to do with the essence of the page content. The classification is also used in face detection and face recognition tasks. Classification is used as a tool for many other tasks:

- removal of homonymy in the processing of natural languages;
- in search engines - to limit the search area in order to improve accuracy (vertical search);
- automatic detection of the language in which the text is written;
- sentiment analysis (determination of the emotional color of the text).

This list can be continued for a long time. For example, in medicine, classification algorithms are used to reconstruct a 3D model of the brain from a series of MRI images, as well as to diagnose patients with Alzheimer's syndrome.

CHAPTER 2
CLASSIFIERS ON THE BASIS OF ARTIFICIAL NEURAL NETWORKS

What is a neural network? To get started, Let's explain a type of artificial neuron called a perceptron. Perceptrons were developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts. Today, it's more common to use other models of artificial neurons - in this book, and in much modern work on neural networks, the main neuron model used is one called the sigmoid neuron. We'll get to sigmoid neurons shortly. But to understand why sigmoid neurons are defined the way they are, it's worth taking the time to first understand perceptrons. So how do perceptrons work? A perceptron takes several binary inputs, x_1, x_2, x_3 and produces a single binary output fig 2.1:

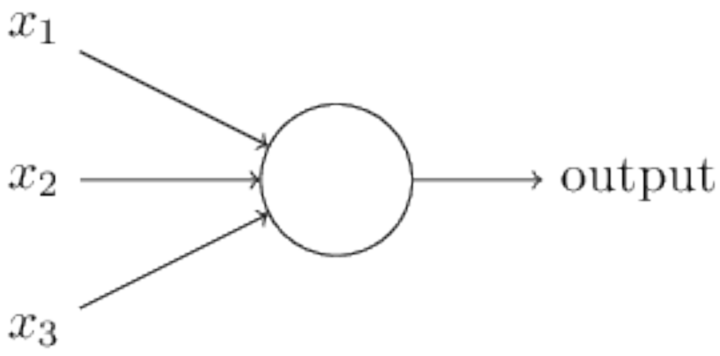


Fig 2.1: Perceptron

In the example shown the perceptron has three inputs, x_1, x_2, x_3. In general it could have more or fewer inputs. Rosenblatt proposed a simple rule to compute the output. He introduced *weights*, $w_1, w_2, \ldots$, real numbers expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum w_j x_j$ is less than or greater than some *threshold value*. Just like the weights, the threshold is a real number which is a parameter of the neuron. To put it in more precise algebraic terms:

$$output = \{0\ if \sum w_j x_j \leq threshold, 1 \text{ if } \sum w_j x_j > threshold$$

That's all there is to how a perceptron works That's the basic mathematical model. A way you can think about the perceptron is that it's a device that makes decisions by weighing up evidence. Let me give an example. It's not a very realistic example, but it's easy to understand, and we'll soon get to more realistic examples. Suppose the weekend is coming up, and you've heard that there's going to be a cheese festival in your city. You like cheese, and are trying to decide whether or not to go to the festival. You might make your decision by weighing up three factors:

1. Is the weather good?
2. Does your boyfriend or girlfriend want to accompany you?
3. Is the festival near public transit? (You don't own a car).

We can represent these three factors by corresponding binary variables x_1, x_2,, and x_3. For instance, we'd have $x_1 = 1$ if the weather is good, and $x_1 = 0$ if the weather is bad. Similarly, $x_2 = 0$ if your boyfriend or girlfriend wants to go, and $x_2 = 0$ if not. And similarly again for x_3 and public transit.

Now, suppose you absolutely adore cheese, so much so that you're happy to go to the festival even if your boyfriend or girlfriend is uninterested and the festival is hard to get to. But perhaps you really loathe bad weather, and there's no way you'd go to the festival if the weather is bad. You can use perceptrons to model this kind of decision-making. One way to do this is to choose a weight $w_1 = 6$ for the weather, and $w_2 = 2$ and $w_3 = 2$ for the other conditions. The larger value of w1w1 indicates that the weather matters a lot to you, much more than whether your boyfriend or girlfriend joins you, or the nearness of public transit. Finally, suppose you choose a threshold of 5 for the perceptron. With these choices, the perceptron implements the desired decision-making model, outputting 11 whenever the weather is good, and 0 whenever the weather is bad. It makes no difference to the output whether your boyfriend or girlfriend wants to go, or whether public transit is nearby.

By varying the weights and the threshold, we can get different models of decision-making. For example, suppose we instead chose a threshold of 3. Then the perceptron would decide that you should go to the festival whenever the weather was good *or* when both the festival was near public transit *and* your boyfriend or girlfriend was willing to join you. In other words, it'd be a different model of decision-making. Dropping the threshold means you're more willing to go to the festival.

Obviously, the perceptron isn't a complete model of human decision-making! But what the example illustrates is how a perceptron can weigh up different kinds of evidence in order to make decisions. And it should seem plausible that a complex network of perceptrons could make quite subtle decisions fig. 2.2:

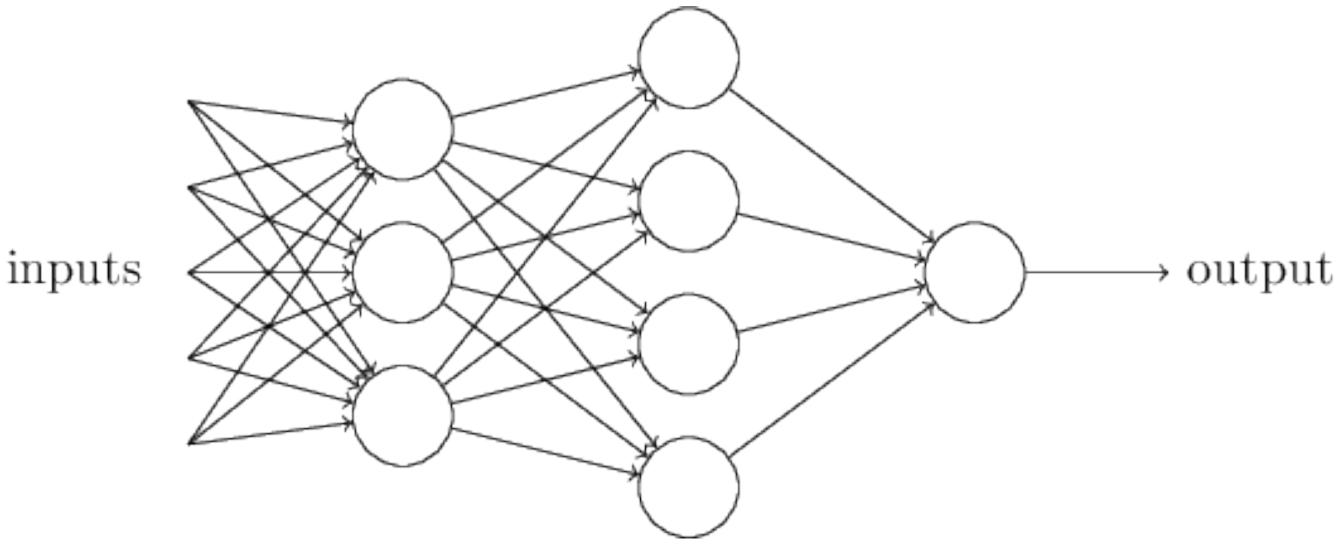


Fig 2.2: Complex network of perceptrons

In this network, the first column of perceptrons - what we'll call the first *layer* of perceptrons - is making three very simple decisions, by weighing the input evidence. What about the perceptrons in the second layer? Each of those perceptrons is making a decision by weighing up the results from the first layer of decision-making. In this way a perceptron in the second layer can make a decision at a more complex and more abstract level than perceptrons in the first layer. And even more complex decisions can be made by the perceptron in the third layer. In this way, a many-layer network of perceptrons can engage in sophisticated decision making.

Incidentally, when I defined perceptrons I said that a perceptron has just a single output. In the network above the perceptrons look like they have multiple outputs. In fact, they're still single output. The multiple output arrows are merely a useful way of indicating that the output from a perceptron is being used as the input to several other perceptrons. It's less unwieldy than drawing a single output line which then splits.

Let's simplify the way we describe perceptrons. The condition $\sum w_j x_j$>threshold is cumbersome, and we can make two notational changes to simplify it. The first change is to write $\sum w_j x_j$ as a dot product, w·x≡$\sum w_j x_j$ where w and x are vectors whose components are the weights and inputs, respectively. The second change is to move the threshold to the other side of the inequality, and to replace it by what's known as the perceptron's *bias*, b≡−thresholdb≡−threshold. Using the bias instead of the threshold, the perceptron rule can be rewritten: output={$0\ if\ w * x + b \leq 0, 1\ if\ w * x + b > 0$.

We can think of the bias as a measure of how easy it is to get the perceptron to output a 11. Or to put it in more biological terms, the bias is a measure of how easy it is to get the perceptron to *fire*. For a perceptron with a really big bias, it's extremely easy for the perceptron to output a 11. But if the bias is very negative, then it's difficult for the perceptron to output a 11. Obviously, introducing the bias is only a small change in how we describe perceptrons, but we'll see later that it leads to further notational simplifications. Because of this, in the remainder of the book we won't use the threshold, we'll always use the bias.

Let's described perceptrons as a method for weighing evidence to make decisions. Another way perceptrons can be used is to compute the elementary logical functions we usually think of as underlying computation, functions such as AND, OR, and NAND. For example, suppose we have a perceptron with two inputs, each with weight −2, and an overall bias of 3. Here's our perceptron fig. 2.3:

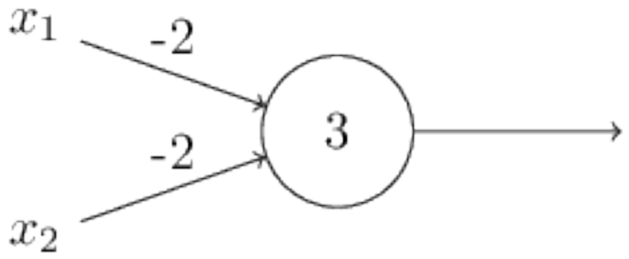


Fig 2.3: Logical perceptron

Then we see that input 00 produces output 1, since (−2)∗0+(−2)∗0+3=3 is positive. Here, I've introduced the ∗ symbol to make the multiplications explicit. Similar calculations show that the inputs 01 and 10 produce output 1. But the input 11 produces output 0, since (−2)∗1+(−2)∗1+3=−1is negative. And so our perceptron implements a NAND gate!

The NAND example shows that we can use perceptrons to compute simple logical functions. In fact, we can use networks of perceptrons to compute *any* logical function at all. The reason is that the NAND gate is universal for computation, that is, we can build any computation up out of NAND gates. For example, we can use NAND gates to build a circuit which adds two bits, x_1 and x_2. This requires computing the bitwise sum, $x_1 \oplus x_2$, as well as a carry bit which is set to 1 when both x_1 and x_2 are 1, i.e., the carry bit is just the bitwise product $x_1 x_2$(fig. 2.4):

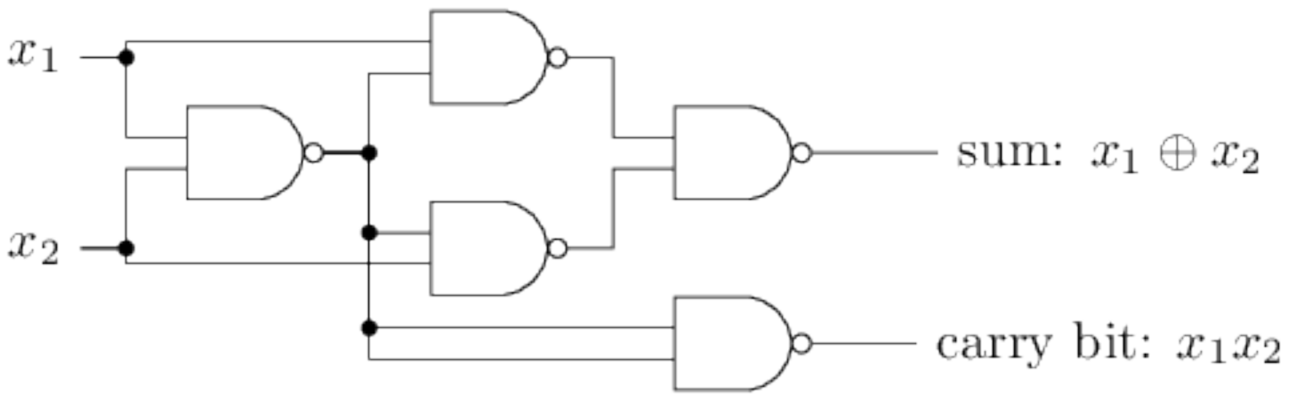


Fig 2.4: Equivalent network

To get an equivalent network of perceptrons we replace all the NAND gates by perceptrons with two inputs, each with weight −2, and an overall bias of 3. Here's the resulting network. Note that we moved the perceptron corresponding to the bottom right NAND gate a little, just to make it easier to draw the arrows on the diagram:

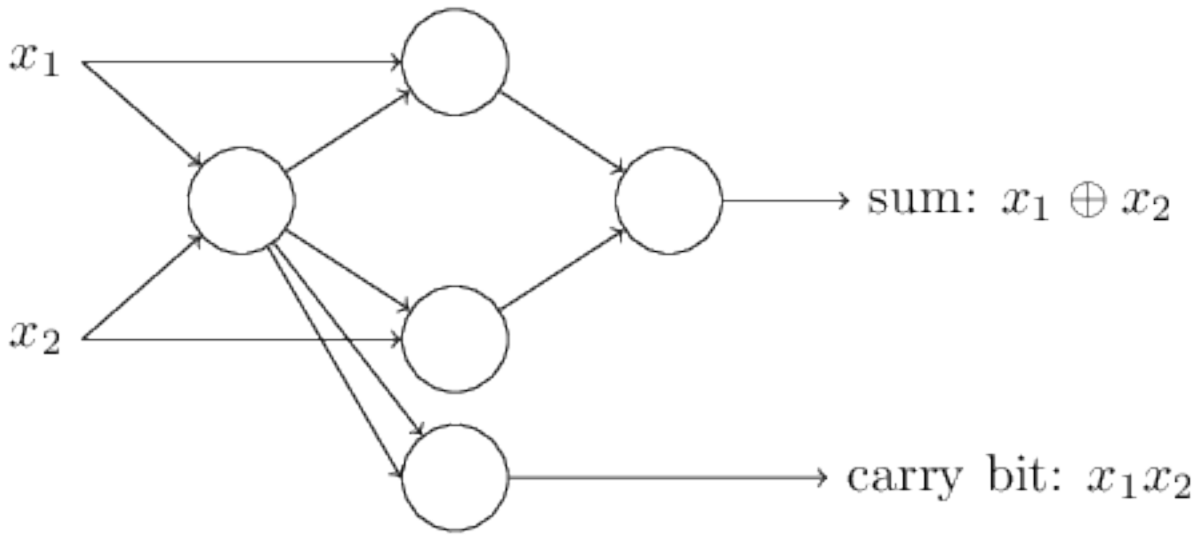


Fig 2.5: Final network

One notable aspect of this network of perceptrons is that the output from the leftmost perceptron is used twice as input to the bottommost perceptron. When I defined the perceptron model I didn't say whether this kind of double-output-to-the-same-place was allowed. Actually, it doesn't much matter. If we don't want to allow this kind of thing, then it's possible to simply merge the two lines, into a single connection with a weight of -4 instead of two connections with -2 weights.

Up to now we been drawing inputs like x1 and x2as variables floating to the left of the network of perceptrons. In fact, it's conventional to draw an extra layer of perceptrons - the *input layer* - to encode the inputs(fig 2.6):

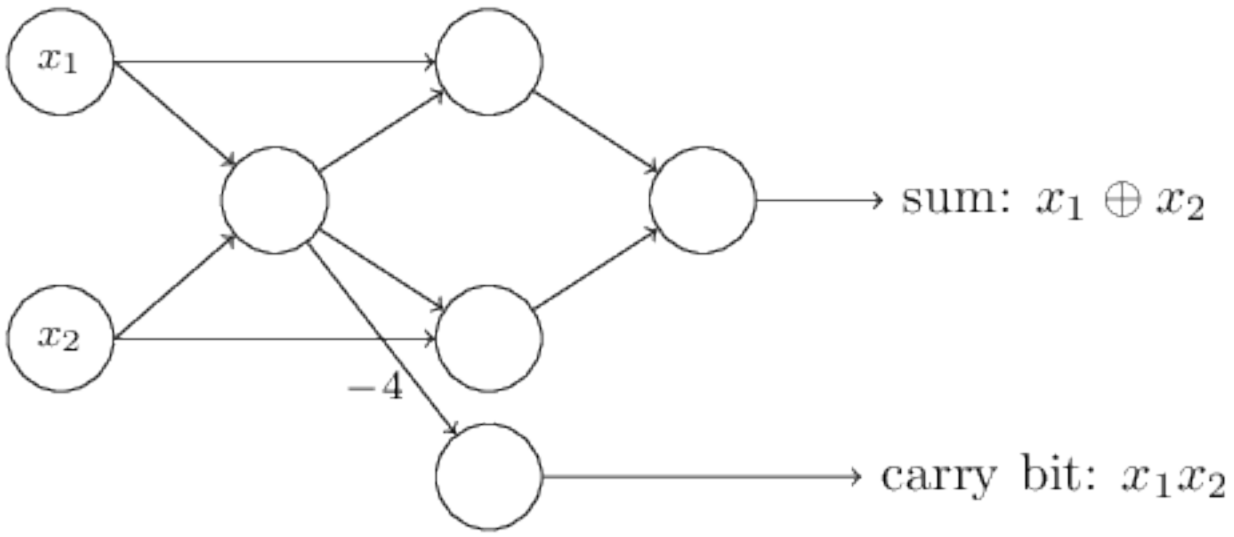


Fig 2.6: Marked network

This notation for input perceptrons, in which we have an output, but no inputs, is a shorthand. It doesn't actually mean a perceptron with no inputs. To see this, suppose we did have a perceptron with no inputs. Then the weighted sum ∑jwjxj would always be zero, and so the perceptron would output 11 if b>0b>0, and 00 if b≤0b≤0. That is, the perceptron would simply output a fixed value, not the desired value (x1x1, in the example above). It's better to think of the input perceptrons as not really being perceptrons at all, but rather special units which are simply defined to output the desired values, x1,x2,….

The adder example demonstrates how a network of perceptrons can be used to simulate a circuit containing many NAND gates. And because NAND gates are universal for computation, it follows that perceptrons are also universal for computation.

The computational universality of perceptrons is simultaneously reassuring and disappointing. It's reassuring because it tells us that networks of perceptrons can be as powerful as any other computing device. But it's also disappointing, because it makes it seem as though perceptrons are merely a new type of NAND gate. That's hardly big news.

However, the situation is better than this view suggests. It turns out that we can devise *learning algorithms* which can automatically tune the weights and biases of a network of artificial neurons. This tuning happens in response to external stimuli, without direct intervention by a programmer. These learning algorithms enable us to use artificial neurons in a way which is radically different to conventional logic gates. Instead of explicitly laying out a circuit of NAND and other gates, our neural networks can simply learn to solve problems, sometimes problems where it would be extremely difficult to directly design a conventional circuit.

2.1. An overview of topologies of neural networks for solving the classification problem

Neural networks can be classified according to the following indicators Table 2.1:

Table 2.1

Classification of neural networks

Nature teaching	Weight adjustment	Input information type	Model network
With a teacher	Fixed	Analogue	Direct distribution
Without a teacher	Dynamic	Binary	Recurrent
With reinforcements	-	-	Kahonen network

Classification of neural networks by the nature of learning:

- Neural networks that use learning with a teacher

Teaching with a teacher assumes that for each input vector there is a target vector representing the desired output. Together they are called a learning pair. Typically, the network learns on a certain set of such learning pairs. The initial vector is initialized, the output of the network is calculated and compared with the corresponding target vector. The scales then change according to the algorithm, which helps minimize the error.

Vectors of the training set are presented sequentially, errors are calculated and weights are adjusted for each vector, until the error in the entire training set does not reach an acceptable level.

- Neural Networks Using Teaching Without Teachers

Learning without a teacher is a more appropriate model of training in terms of biological neural networks. Such training does not require a target vector for outputs and, accordingly, does not require comparison with known loyal responses. The training set consists of only the input vectors. The learning algorithm adjusts the weight of the network so that the agreed output vectors are obtained, that is, the presentation of sufficiently close input vectors gives the same outputs. The learning process highlights the statistical properties of the learning set and groups similar vectors into classes.

- Neural networks using reinforcement training

Learning with reinforcements is one of the ways of machine learning, during which the system learns while interacting with a particular environment. The feedback of the environment on the decisions made are reinforcement signals, so this training is a special case of teaching with a teacher. Some reinforcement rules are based on implicit teachers, which can be attributed to non-teacher education.

- Adjustment of weight coefficients
    - Fixed-line networks - Weighted coefficients of the neural network are determined by certain values at once, based on the conditions of the task.
    - Networks with dynamic connections - for them the adjustment of synaptic weights occurs during the learning process.
- Input information type
    - Analogue - the input information is presented in the form of valid numbers.

    - Binary - all incoming information in such networks is represented in the form of zeros and units.
- Model of neural networks for solving the classification problem
    - Net distribution networks - all connections and flow of processing are directed from the input neurons to the output. These networks include, for example: the simplest perceptron Rosenblatt and a multi-layer perceptron.
    - Reverse distribution networks (recurrent) - the signal from the source neurons or the neurons of the hidden layer is partially transmitted back to the inputs of the neurons of the previous layer.
    - Self-organized maps or Kohonen Networks - • A network class, as a rule, learns without a teacher and is successfully applied in recognition tasks. Networks of this class are capable of detecting novelty in the input: if, after learning, the network will encounter a set of data that is unlike any known model, then it will not be able to classify such a set and thereby reveal its novelty. The Kohonen network has only two layers: input and output.
    - Fuzzy neural networks – are trained on the basis of a classified sample. Adjust the value of membership functions in the learning process using the inverse spread method of the error. This type of network includes: Nefclass, Nefprox, Anfis, TSK.

For his research, the author of this work used a fuzzy classifier based on the TSK network. This network has a multilayer structure with a direct signal propagation, the value of the output of the network can be changed by adjusting the parameters of the layer elements, which allows using the error propagation algorithm

2.2. Fuzzy neural networks and its features. Membership functions

Neural networks are convenient for image recognition tasks, but very uncomfortable to explain how they implement it. They can automatically acquire knowledge, but the process of their learning is often slow enough, and the analysis of the trained network is very complicated (a trained network is usually a "black screen" for the user). At the same time, it is difficult to enter any prior information (expert knowledge) to accelerate the learning process in the neural network.

Systems with fuzzy logic, on the contrary, are convenient to explain the conclusions obtained with their help, but they cannot automatically acquire knowledge for their use in the mechanisms of outputs. The need to split universal sets into separate areas, as a rule, limits the number of input variables in such systems to a small value.

The direct distribution neural network can approximate any system that is based on fuzzy rules, and any neural network of direct distribution can be approximated by a system based on fuzzy rules.

The neuro-fuzzy network is a representation of a fuzzy output system in the form of a neural network that is convenient for learning, analysis and use. The structure of the neuro-fuzzy network corresponds to the main blocks of fuzzy output systems.

The main properties of neuro-fuzzy networks is that:

- neuro-fuzzy networks are based on fuzzy systems that are learned using methods used in neural networks;
- a neuro-fuzzy network is usually a multi-layer (often triple) neural network. The first layer is the input variables, the average is fuzzy rules, and the third is the output variables. The weight of the connection corresponds to the fuzzy sets of input and output variables. Sometimes a five-layer architecture is used. In the general case, a fuzzy system does not necessarily have to be presented in this form, but it is a convenient model for the application of teaching methods;
- neuro-fuzzy network always (before, during and after training) can be interpreted as a system of fuzzy rules;
- the learning procedure takes into account the semantic properties of the fuzzy system. This is manifested in the limitation of the possible modifications that apply to the parameters that are being adjusted. It is necessary, however, to say that not all methods of teaching neuro-fuzzy networks take into account the semantics of the system;

Types of combination of fuzzy logic and neural networks by way of interaction distinguish the following:

- fuzzy neural systems. In this case, neural networks use the principles of fuzzy logic to accelerate the process of debugging or improve other parameters. With this approach, fuzzy logic is just a tool of neural networks, and such a system cannot be interpreted in fuzzy rules, since it is a "black chest";

- competing neuro-fuzzy systems. In such models, the fuzzy system and the neural network work on one task, without affecting the parameters of each other. It is possible to sequentially process data first by one system, then another;
- parallel neuro-fuzzy systems. In such systems, debugging of parameters is performed with the help of neural networks. Then the fuzzy system functions independently. The following types of parallel neuro-fuzzy models are distinguished:
- Integrated (hybrid) neuro-fuzzy systems with close interaction of fuzzy logic and neural networks.

The term "neuro-fuzzy networks" most often refers to systems of this type. Typically, integrated systems are Mandani or Takagi-Sugeno systems.

By way of displaying fuzzy sets in the network structure, neuro-fuzzy networks have three main types:

- systems built on random fuzzy sets. In such systems, the degree of affiliation is described only for some of the values in the definition area and the function of membership is presented as a vector. Each degree of affiliation corresponds to only one input or output neuron. There are two approaches to implementing such systems. In the first system, it simply approximates the correspondence of the outputs to the inputs, such a system is a "black chest". The second creates a system with a special architecture, in which fuzzy rules are implemented;
- systems, parameterized functions whose belongings are stored in the neurons. An example of such systems is ANFIS (Adaptive-Network-based Fuzzy Inference System);
- systems in which parametric functionality of belonging is used as the weight of connections between neurons. Such a system can otherwise be called a perceptron with fuzzy connections or fuzzy perceptron.

By the nature of training distinguish the following types of neuro-fuzzy networks:

- self-adjusting neuro-fuzzy networks - with the adaptation of structure and parameters;
- adaptive neuro-fuzzy networks - with rigid structure and adaptation of network parameters.

Adaptive neuro-fuzzy networks by the type of optimization method are divided into those using deterministic methods such as gradient search, and those that use stochastic methods, in particular evolutionary ones.

Adaptive neuro-fuzzy networks by type of adaptation parameters are divided on the network with the adaptation of the parameters of the functions of affiliation, networks with the adaptation of the weight of the rules and the network with the adaptation of the parameters of the aggregation operator.

Fuzzy sets have become popular thanks to membership features that add flexibility when processing large amounts of data and its scalability. Consider the basic membership functions that can be used in neuro-fuzzy networks.

Bulk-linear membership functions:

As the first type of membership functions we consider functions that, as their names indicate, consist of lines of straight lines, forming a continuous or piecewise-continuous function. The most typical example of such functions is the "triangular" (fig. 2.7) and "trapezoidal" (fig. 2.8) Of the membership function.

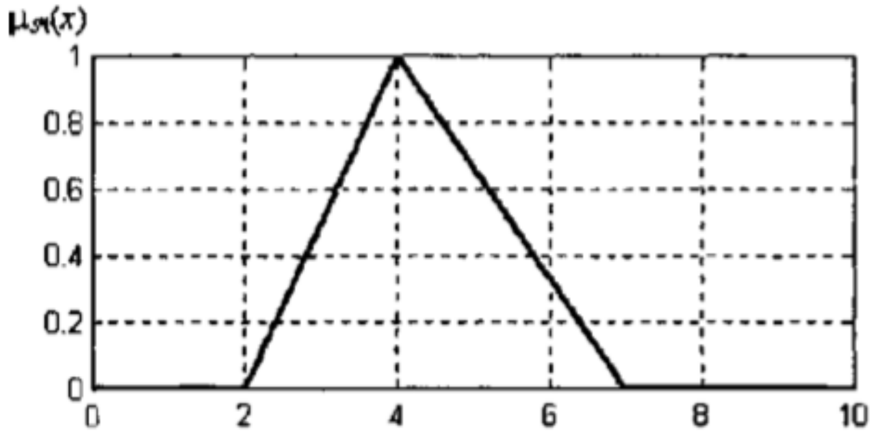


Fig 2.7: Triangular membership function

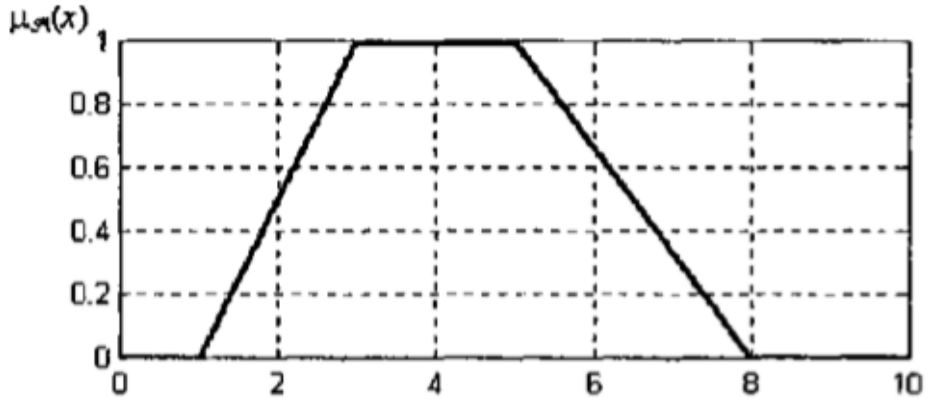


Fig 2.8: Trapezoid function of membership

The first of these membership functions in the general case can be given analytically by the following expression:

$$f_{\Delta}(x;,a,b,c) = \left\{0, x \leq a\frac{x-a}{b-a}, a \leq x \leq b\frac{c-x}{c-b}, b \leq c \leq x \;\; 0, \qquad c \leq x \right\} \qquad (2.1),$$

where a, b, c are some numerical parameters that take arbitrary real values and are ordered by the ratio: $a \leq b \leq c$.

The trapezoidal membership function in the general case can be given analytically by the following expression:

$$f_T(x;a,b,c,d) = \{0, x \leq a\frac{x-a}{b-a} a \leq x \leq b, b \leq x \leq c\frac{d-x}{d-c}, c \leq x \leq d, d \leq x\} \qquad (2.2),$$

where a, b, c, d are some numerical parameters that take arbitrary real values and arranged by the ratio: $a \leq b \leq c \leq d$.

These functions are used for the task of such properties of sets that characterize the uncertainty of type: "approximately equals", "average value", "located in interval", "similar object", "similar to object", etc. They also serve to represent fuzzy numbers and intervals, which will be discussed further.

Π- similar membership functions

To this type of membership functions can be attributed a whole class of curves, which in their shape resemble a bell, a smooth trapeze or the letter "P".

The first of these functions is called - the P-image function, and in the general case is given analytically by the following expression:

$$\boldsymbol{f_{\Pi}(x;,a,b,c,d) = f_s(x;a,b) * f_z(x;c,d)} \qquad (2.3),$$

where a, b, c, d are some numerical parameters that take arbitrary real values and are arranged by the ratio: $a \le b \le c \le d$, and the sign "*" denotes the usual arithmetic product of the values of the corresponding functions. This function is shown in fig 2.8

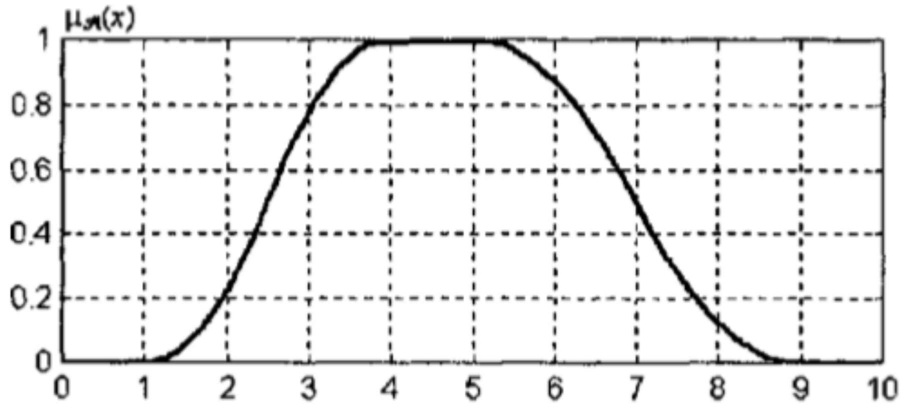


Fig 2.8: P-shaped membership function

To the P-shaped functions also refers to the so-called bell-like function (fig. 2.8.), Which in general is given analytically by the following expression:

$$\boldsymbol{f_{\Pi 2}(x;,a,b,c,d) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}} \qquad (2.4),$$

where a, b, c are some numerical parameters that take arbitrary real values and are arranged by the ratio: a <b <s, with the parameter b> 0. Here the function | x | denotes the module of the real number.

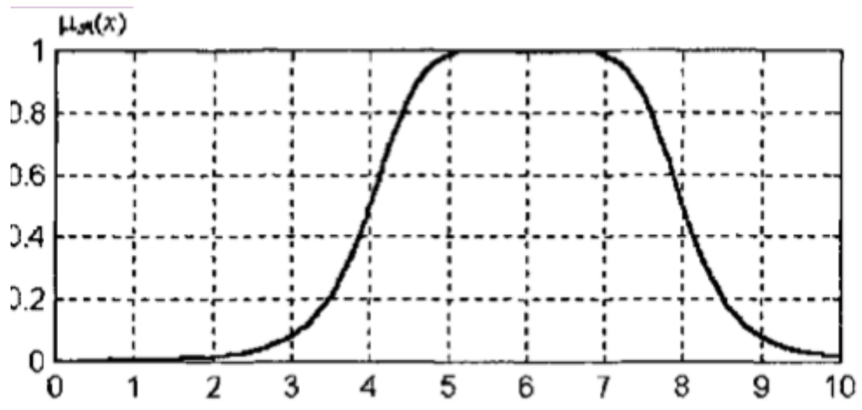


Fig 2.9: Bell-like functions

Finally, the last of the functions under consideration of this type is well known in the theory of probabilities the function of the density of normal distribution in the assumption that $\sqrt{2\pi\sigma} = 1$ and which in our case is given analytically by the following expression:

$$\boldsymbol{f}_{\Pi 3}(\boldsymbol{x};\boldsymbol{\sigma},\boldsymbol{c}) = \boldsymbol{e}^{\frac{-(\boldsymbol{x}-\boldsymbol{c})^2}{2\boldsymbol{\sigma}^2}} \qquad (2.5),$$

where σ *and* c - numerical parameters, with the square of the first of them σ^2 in the theory of probabilities is called the dispersion of the distribution, and the second parameter c is the mathematical expectation.

Obviously, these last types of membership functions generate normal convex fuzzy sets (fig. 2.10).

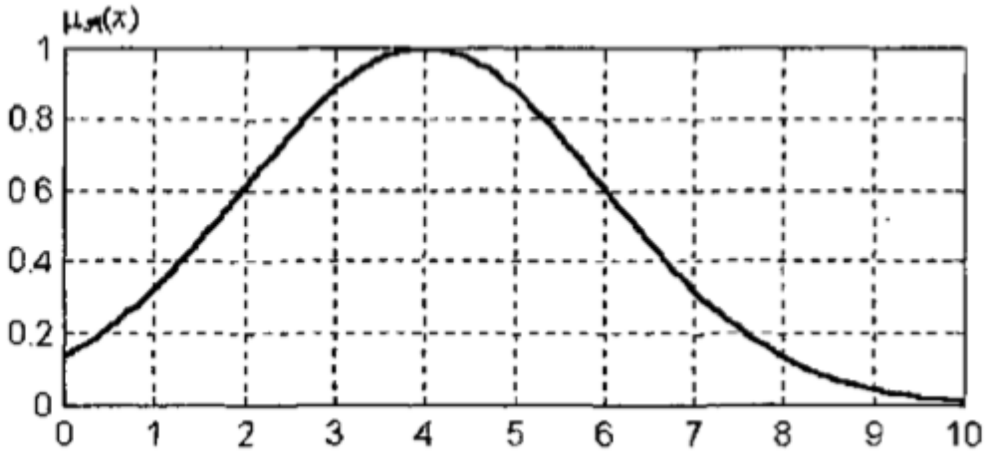


Fig 2.10: Normal distribution function

These membership functions are used most often in the construction of fuzzy neural networks. In this paper, the author considers the triangular and sigmoidal membership function, to create a hybrid network. The choice has fallen to these functions due to their simplicity and speed of calculation on personal computers.

2.3. Network TSK. Topology Learning algorithm

Figure 2.11 depicts the structure of the fuzzy neural network TSK. In this network, nesting characteristics, properties, seductiveness and functional elements are distinguished:

1) Teaching is conducted with a teacher. That is, for each input vector there is an expected output value.

2) The sample is divided into 2 parts: training and training.

The generalized output schema in the TSK model when using rules and variables can be presented as follows:

$$R_1: if\ x_1 \in A_1^{(1)};\ x_2 \in A_2^{(1)}, \dots, x_n \in A_n^{(1)}, then\ y_1 = p_{10} + \sum_{j=1}^{N} p_{1j}x_j,$$

$$R_M: if\ x_1 \in A_1^{(M)};\ x_2 \in A_2^{(M)}, \dots, x_n \in A_n^{(M)}, then\ y_M = p_{M0} + \sum_{j=1}^{N} p_{Mj}x_{j,}$$

where $A_n^{(M)}$ - the meaning of a linguistic variable x_i for the rule R_M with membership function (FN)

$$\mu_A^{(k)}(x_i) = \frac{1}{1+\left(\frac{x_i - c_i^{(k)}}{\sigma_i^{(k)}}\right)^{2b_i^{(k)}}}, i = \underline{1,N}, k = \underline{1,M},$$

In the fuzzy neural network TSK cross-sectional rules R_k FN is given in the form and product:

$$\mu_A^{(k)}(x) = \frac{\prod_{j=1}^{N} \quad 1}{1+\left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}}$$

With the exit rules, the composition of the output results of the network is presented by the following formula:

$$y(x) = \frac{\sum_{k=1}^{M} \omega_k y_k(x)}{\sum_{k=1}^{M} \omega_k}$$

where:

$$y_k(x) = p_{k0} + \sum_{j=1}^{N} p_{kj} x_j \, , \quad \omega_k = \mu_A^{(k)}(x)$$

Fuzzy neuron network TSK is given by a multilayered structural network, as shown in fig. 2.11. In this network, there are 5 layers.

1. The first layer performs the separate phasing of each variable x_i, $i = \overline{1, N}$, defining for each k rules for output values MF $\mu_A^{(k)}(x_i)$ according to the function of phasing. This is a parametric parameter layer $c_j^{(k)}, \ \sigma_j^{(k)}, \ b_j^{(k)}$, which are subject to adaptation in the learning process.
2. The second layer performs the aggregation of individual variables x_i, determining the resulting degree of affiliation $\omega_k = \mu_A^{(k)}(x)$ for the vector *x* conditions k rules. This is not a parametric layer.
3. The third layer is a TSK function generator, which calculates the value

$$y_k(x) = p_{k0} + \sum_{j=1}^{N} p_{kj} x_j \, .$$

This layer also has a multiplication function $y_k(x)$ on w_k, formed on the previous layer. This is a parametric layer in which the linear parameters are subject to adaptation, p_{k0}, p_{kj} for $i = \underline{1, N}, k = \underline{1, M}$, defining the function of the consequences of the rules.

4. The fourth layer consists of 2 neurons-adder, one of which calculates a weighted sum of signals $y_k(x)$, and the second determines the amount of weight

$$\sum_{k=1}^{M} \omega_k .$$

5. The fifth layer consists of one single neuron. In it the weight is subject to normalization and the output signal is calculated in accordance with the expression

$$y(x) = \frac{f_1}{f_2} = \frac{\sum_{k=1}^{M} \omega_k y_k(x)}{\sum_{k=1}^{M} \omega_k}$$

It is also not a parametric layer..

From the above description it follows that the fuzzy network TSK contains only 2 parametric layers (first and third), whose parameters are specified in the learning process. Parameters of the first layer ($c_j^{(k)},\ \sigma_j^{(k)},\ b_j^{(k)}$) will be called nonlinear, and the parameters of the third layer - linear weights.

The general expression for the functional dependence on the TSK network is given as follows:

$$y(x) = \frac{1}{\sum_{k=1}^{M} \prod_{j=1}^{M} \mu_A^k(x_j)} \sum_{k=1}^{M} (p_{k0} + \sum_{j=1}^{N} p_{kj} x_j) \prod_{j=1}^{N} \mu_A^{(k)}(x_j)$$

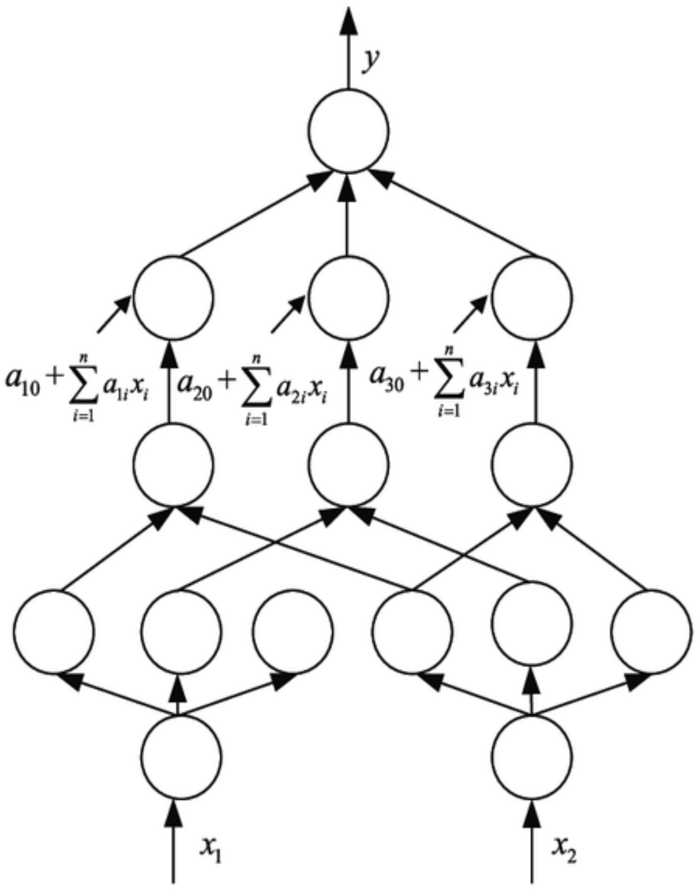

y
$a_{10} + \sum_{i=1}^{n} a_{1i} x_i$
$a_{20} + \sum_{i=1}^{n} a_{2i} x_i$
$a_{30} + \sum_{i=1}^{n} a_{3i} x_i$
x_1
x_2

Fig 2.11: Takagi-Sugeno-Kang structure

2.4. Algorithm for starting a TSK network

Consider the hybrid learning algorithm. In the hybrid algorithm, the parameters to be adapted are divided into 2 groups. The first one consists of the linear parameters of the third layer, and the second group - from the parameters of the nonlinear FN of the first layer. Clarification of the parameters takes place in 2 stages.

In the first stage, when fixing certain values of the parameters of the membership function, solving the system of linear equations, the linear parameters of the polynomial TSK are calculated. With known values of the FP, the dependence for the output can be represented as a linear form with respect to the parameter p_{kj}:

$$y_k(x) = \sum_{k=1}^{M} \omega_k'(p_{k0} + \sum_{j=1}^{N} p_{kj} x_j) \text{, where } \omega_k' = \frac{\prod_{j=1}^{N} \mu_A^{(k)}(x_j)}{\sum_{r=1}^{M} \prod_{j=1}^{N} \mu_A^{(r)}(x_j)}, \quad k = \overline{1,M}$$

With the dimension of the training sample $L(x^{(l)}, d^{(l)})$, $(l = 1,2,\dots,L)$ and replacing the output signal with the expected value of the network $d^{(l)}$ get a system with L linear equations of the form[14]

$$\begin{bmatrix} \omega_{11}' & \omega_{11}'x_1^{(1)} \dots \omega_{11}'x_N^{(1)} \dots \omega_{1M}' & \omega_{1M}'x_1^{(1)} \dots \omega_{1M}'x_N^{(1)} \\ \omega_{21}' & \omega_{21}'x_1^{(2)} \dots \omega_{21}'x_N^{(2)} \dots \omega_{2M}' & \omega_{2M}'x_1^{(2)} \dots \omega_{2M}'x_N^{(2)} \\ \dots & \dots \quad \dots \quad \dots & \\ \omega_{L1}' & \omega_{L1}'x_1^{(L)} \dots \omega_{L1}'x_N^{(L)} \dots \omega_{LM}' & \omega_{LM}'x_1^{(L)} \dots \omega_{LM}'x_N^{(L)} \end{bmatrix} \times \begin{bmatrix} p_{10} \\ p_{11} \\ \dots \\ p_{1N} \\ \dots \\ p_{M0} \\ P_{M1} \\ \dots \\ p_{MN} \end{bmatrix} = \begin{bmatrix} d^{(1)} \\ d^{(2)} \\ \dots \\ d^{(L)} \end{bmatrix}$$

where ω_{il}' means the activation level (weight) of the condition i rules upon presentation l input vector x^l. This statement can be written in matrix form

$$Ap = d$$

Dimension matrix A equates to $L(N+1)M$. This number of rows L Usually there are significantly more columns $(N+1)M$. The solution of this system of equations can be obtained both by ordinary methods, and in one step, using the pseudo inversion of the matrix A:

$$p = A^{+}d,$$

where A^{+} - pseudo inverse matrix.

In the second stage after fixing the value of the linear parameters p_{kj} the actual output signals are calculated$y^{(l)}$, $l = 1,2,...L$, for this use the linear dependence:

$$y^{(L)} = Ap.$$

After that, the error vector is calculated $\varepsilon = y - d$ and criterion

$$E = \frac{1}{2}\sum_{l=1}^{L}(y(x^{(l)}) - d^{(l)})^2$$

Error signals are sent through the network in reverse order according to the Back-Propagation method up to the first layer, where the components can be calculated. $c_j^{(k)},\quad \sigma_j^{(k)},\quad b_j^{(k)}$. After calculating the gradient vector, the gradient method is started. The corresponding training formulas (for the simplest method of rapid descent) take the form:

$$c_j^{(k)}(n+1) = c_j^{(k)}(n) - \eta_c \frac{\partial E(n)}{\partial c_j^{(k)}},$$

$$\sigma_j^{(k)}(n+1) = \sigma_j^{(k)}(n) - \eta_\sigma \frac{\partial E(n)}{\partial \sigma_j^{(k)}},$$

$$b_j^{(k)}(n+1) = b_j^{(k)}(n) - \eta_b \frac{\partial E(n)}{\partial b_j^{(k)}},$$

where n - iteratoion number.

After refining the nonlinear parameters, the process of adaptation of the linear parameters of the function TSK (first stage) and nonlinear parameters (second stage) is restarted. This cycle continues until all process parameters are stabilized.

When familiarizing with the network, the bell-like membership function was used. corresponding formulas of the gradient target function method for one pair of data (x,d) take a look[14]:

$$\frac{\partial E}{\partial c_j^{(k)}} = (y(x)-d)\sum_{r=1}^{M}(p_{r0} + \sum_{j=1}^{N} p_{rj}x_j)\cdot\frac{\partial w_r'}{\partial c_j^{(k)}},$$

$$\frac{\partial E}{\partial \sigma_j^{(k)}} = (y(x)-d)\sum_{r=1}^{M}(p_{r0} + \sum_{j=1}^{N} p_{rj}x_j)\cdot\frac{\partial w_r'}{\partial \sigma_j^{(k)}},$$

$$\frac{\partial E}{\partial b_j^{(k)}} = (y(x)-d)\sum_{r=1}^{M}(p_{r0} + \sum_{j=1}^{N} p_{rj}x_j)\cdot\frac{\partial w_r'}{\partial b_j^{(k)}}$$

corresponding derivatives

$$\frac{\partial w_r'}{\partial c_j^{(k)}},\quad \frac{\partial w_r'}{\partial \sigma_j^{(k)}},\quad \frac{\partial w_r'}{\partial b_j^{(k)}}$$

take the following form:

$$\frac{\partial w_r'}{\partial c_j^{(k)}} = \frac{\delta_{rk}m(x_j)-l(x_j)}{[m(x_j)]^2}\prod_{i=1,i\neq j}^{M}\frac{\left[\frac{2b_j^{(k)}}{\sigma_j^{(k)}}\left(\frac{x_j-c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}-1}\right]}{\left[1+\left(\frac{x_j-c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}\right]^2}\mu_A^{(k)}(x_i),$$

$$\frac{\partial w_r'}{\partial \sigma_j^{(k)}} = \frac{\delta_{rk}m(x_j)-l(x_j)}{[m(x_j)]^2}\prod_{i=1,i\neq j}^{M}\frac{\left[\frac{2b_j^{(k)}}{\sigma_j^{(k)}}\left(\frac{x_j-c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}\right]}{\left[1+\left(\frac{x_j-c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}\right]^2}\mu_A^{(k)}(x_i),$$

$$\frac{\partial w_r'}{\partial b_j^{(k)}} = \frac{\delta_{rk} m(x_j) - l(x_j)}{[m(x_j)]^2} \prod_{i=1, i \neq j}^{M} \frac{\left[-2\left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}} \ln\left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)\right]}{\left[1 + \left(\frac{x_j - c_j^{(k)}}{\sigma_j^{(k)}}\right)^{2b_j^{(k)}}\right]^2} \mu_A^{(k)}(x_i)$$

for $r = 1,2...M$,

where δ_{rk} - Kronecker Delta,

$$l(x_j) = \prod_{j=1}^{N} \mu_A^{(k)}(x_j) \; ; \quad m(x_j) = \sum_{k=1}^{M} \prod_{j=1}^{N} \mu_A^{(k)}(x_j)$$

In the practical implementation of the hybrid method of learning fuzzy networks, the dominant factor of their adaptation is considered the first stage in which weights are selected using pseudo-inversion in one step. To balance its effects, the second stage is repeated many times in each cycle.

Conclusions: The presented hybrid algorithm is one of the most effective ways of learning fuzzy neural networks. Its characteristic feature is the division of the process into two separate stages in time. Given that the calculated complexity of each optimization algorithm is nonlinearly dependent on the number of parameters to be optimized, the reduction of the dimension of optimization tasks significantly reduces the amount of settlement operations and increases the rate of convergence of the algorithm. Thanks to this, the hybrid algorithm is more effective compared to the usual gradient method.

CHAPTER 3
NEURAL NETWORK ENSEMBLES, REVIEW OF EXISTING ALGORITHMS TO IMPROVE NETWORK RESULTS

3.1. Ensemble of neural network algorithms

Among the existing methods of classifying security events, security incidents, threats, etc., intelligent information technologies are widely used - artificial neural networks (ANNs), fuzzy and neuro-fuzzy systems, evolutionary algorithms, multi-agent and immune systems. However, often due to the complexity of the problem, poor quality of training data and other reasons, it is not possible to achieve a satisfactory quality of the model. Then it makes sense to apply a set of models used together to solve a single problem. This set of models is called a model ensemble (committee). The analysis of the practical use of such systems allows us to assert that increasing the efficiency of their application is possible due to the use of several technologies within the framework of one information security (IS) system, for example, a collective (ensemble) of neural networks. An improved integrated approach to the construction of a neural network ensemble is proposed, which improves the efficiency of solving the problem of classifying security events.

In [1], it is noted that a promising direction for improving ANN is the unification (composition) of many separate ANNs into one system (associative machine). In this case, the errors of individual classification algorithms are mutually compensated. The ensemble organization is considered in a number of works [2–9]. The paper [8] experimentally proved the effectiveness of using the ensemble organization for image recognition. When constructing an ensemble of neural networks, a finite set of pre-trained neural networks are simultaneously used, the output signals of which are combined into a combined estimate that is superior in quality to the results obtained using local networks included in the ensembles.

The ensemble $H(\bar{x})$ of models $h_i(\bar{x})(\mathrm{i} = 1, 2, \ldots, \mathrm{N})$ is a composition of algorithmic operators h_i: $R_d \to R$ and a correcting operation F: RN → R, in which the set of estimates h1 (̅x) , h2 (̅x), ..., hN (̅x), the final estimate H (x) [4] is put into correspondence: H (̅x) = F (h1 (̅x), h2 (̅x),…, hN (̅x)). (1) As you know, the fundamental task in the construction of ensembles is to generate the diversity of the ensemble (or differences in individual models) [6]. It is obvious that the aggregation of similar models in an ensemble cannot lead to a significant increase in the quality of the problem solution. An ensemble of models may be better than individual models included in an ensemble for the following reasons [8]:

- The ensemble reduces the mean square error. The use of an ensemble of models averages the error of each individual model and reduces the influence of instabilities and randomness in the formation of hypotheses. The solution of classification and regression problems is a search for hypotheses about the properties of the system or the next state of the system. If you use a sufficiently large number of models trained on approximately the same set of examples, then you can reduce the instability and randomness of the result by combining the results. Averaging over multiple models based on independent training sets always decreases the expected mean square error.
- Ensembles of models trained on different subsets of the original data have a greater chance of finding the global optimum, as they look for it from different starting points.
- The combined hypothesis may not be in the set of possible hypotheses for the base classifiers, i.e. when constructing a combined hypothesis, the set of possible hypotheses expands. Several approaches are used to build ensembles of models. Most often, the ensemble consists of basic models of the same type, which are trained on different sets of training samples (fig. 3.1).

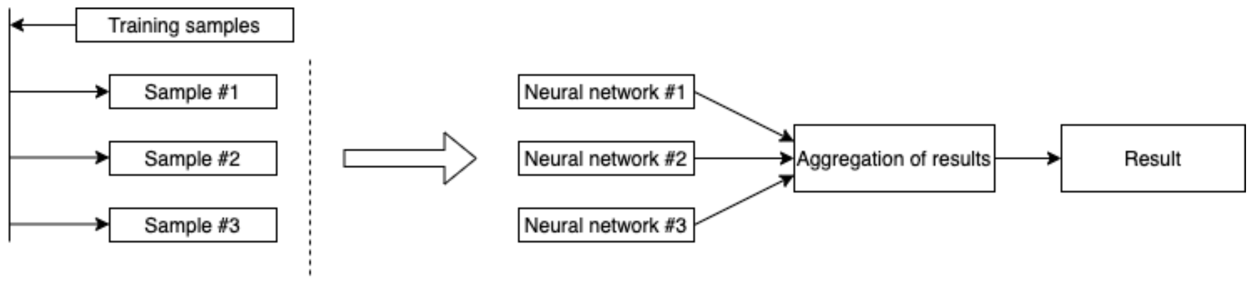


Fig 3.1: General ensemble structure

For the formation of the output value of the ensemble at certain states of the outputs of the models, the following algorithms are most common [11]:

1. Voting. Used in classification tasks. The class is chosen by the simple majority of the ensemble models.

2. Weighted voting. It differs from simple voting by assigning weights (points) to the results of different models. The points take into account the accuracy of the work of different classifiers.

3. Averaging (weighted or unweighted). It is used when solving a regression problem with the help of an ensemble, when the outputs of the models are numeric. The output of the entire ensemble can be defined as a simple average of the outputs of all models. If weighted averaging is performed, then the model outputs are multiplied by the corresponding weights.

4. A mixture of experts. In this case, the weighting factor is a function of the input vector.

The simplest example of a vote is a simple vote:

$$\boldsymbol{Y(x) = F\big(y_1(x), y_2(x), \dots, y_k(x)\big) = \frac{1}{m}\sum_{i=1}^{m} y_i(x),}$$

where x = (x1, x2,…, xk) – input vector;

k - is the number of inputs of the neural network;

$Y(x) = (y_1, y_2, \dots, y_k)$ - is the vector of values of the output signal of the neural network ensemble, F is the function for obtaining the resulting solution. Simple voting is a special case of weighted voting:

$$\boldsymbol{Y(x) = F\big(y_1(x), y_2(x), \dots, y_k(x)\big) = \frac{1}{m}\sum_{i=1}^{m} a_i y_i(x), \sum_{i=1}^{m} a_i = 1, a_i \geq 0}$$

where a_i – weighting factor of the i-th model.

Weighted voting is a special case of a mixture of experts:

$$\boldsymbol{Y(x) = F\big(y_1(x), y_2(x), \dots, y_k(x)\big) = \frac{1}{m}\sum_{i=1}^{m} a_i(x) y_i(x), \sum_{i=1}^{m} a_i(x) = 1, \forall x \in X}$$

In the general case, the solution of the classification problem based on the ANN team consists, firstly, in the formation and training of a finite set of ANNs participating in the solution, and secondly, in determining such a way of coordinating individual solutions so that the final solution is the best. At the next stage, those ANNs are selected from the general pool, with the help of which the final solution will be formed. In the general case, the final solution is a certain function, the input parameters of which are the particular solutions of the ANN included in the ensemble: $R = f(r_1, r_2, \dots, r_n)$ where R is the general solution, ri is the individual solution i -th ANN, n is the total number of ANNs in the ensemble. The function f defines a way to generalize individual solutions. The solution R in the classification problem consists in choosing the number of one of the classes $A_j, j = 1,2, \dots J$, or choosing an empty set in case of rejection of the classification. Each particular decision ri can take on the value of an image or be an empty set if the pattern does not belong to the area of competence of a particular classifier. The scope of competence is understood as a subset of objects of the feature space, within which the scope of a particular classifier with a given subset of recognizable patterns is determined. In the general case, the synthesis of the function f is the central task of using neural network ensembles [12]. Each of the r_i decisions can be assigned a weight and an area of competence defined. The collective decision is determined by the set of individual decisions ri, which belong to the area of competence K (r_i). Thus, the decision of the team is determined by a set of individual decisions corresponding to their areas of competence. This approach contains ANNs whose solutions correspond to the area of competence. At the next stage, the output value of the ensemble is calculated using one of the above algorithms. Currently, the most developed methods for constructing neural network committees are: equal or unequal voting for classification problems and simple or weighted averaging for regression problems [13]. Analysis of existing approaches showed that they do not always provide the required quality of the final decision (accuracy and validity). The most significant disadvantages include:

- the dependence of the final result on the reliability of the determination of the competence coefficients, which can lead to an incorrect result;
- often the training set contains noise emissions, leading to an increase in the probability of erroneous decisions by private classifiers, since an attempt by the training algorithm to tune in to noise degrades the approximating capabilities of the network;

- the need to use a large number of examples of the training sample for the successful implementation of the algorithm;
- the architecture of a neural network ensemble is often redundant, which does not increase the accuracy of solving classification problems and leads to a significant increase in the required computing resource.

An integrated approach to solving classification problems based on neural network ensembles (neural network committees) To improve the quality of obtaining a solution based on a neural network ensemble, an approach is proposed that allows, on the one hand, to reduce the computational costs of implementing the work of a neural network ensemble, and on the other hand, to improve the quality solving classification problems by applying the method of adaptive reduction of the neural network ensemble (selection of the best classifiers based on assessing the degree of compliance with the area of competence of a private neural network classifier and assessing the convergence of the results of private classifiers), as well as choosing and justifying the method for aggregating results (composition of outputs of private classifiers) The task of classification is to determine its belonging to a certain image by the quantitative characteristics of an unknown object. The approach includes a set of stages:

1. Formation of the initial set (pool) of neural networks included in the ensemble (determination of the number of hidden layers of neurons, activation functions, the size of the training sample).

2. Reduction of the neural network ensemble (selection of the best classifiers based on calculating the classification reliability coefficients, comparing the obtained coefficient values with the specified threshold values and assessing the convergence of the results obtained by private classifiers).

3. Selection and justification of the method for aggregating the results (voting method). At the first stage, neural networks are selected (generated) that meet the specified requirements. The architecture and parameters of an artificial neural network - classifier, the size of the training sample are determined. In this case, neural networks are formed in the form of simpler structures, in contrast to the traditional approach to the synthesis of a classifier using a single neural network. Simple networks are fairly easy to learn and less

prone to overfitting. At the second stage, the best classifiers are selected. A special algorithm (referee) is used to assess the competence of the classifier. The competence of a classifier in a given area of the space of representation of objects of classification is understood as its accuracy, i.e. probability of correct classification of objects, whose description belongs to this area. To formalize the method of aggregating results (voting scheme), the coefficient is used μij ≤ 1 the reliability of the classification by the private classifier. The coefficient μij is the proportion of objects with a given value of the image j that fall within the competence of the i-th classifier $\boldsymbol{\mu_{ij}} = \frac{\boldsymbol{F_i(j)}}{\boldsymbol{F(j)}}$, where F(j) – the cumulative decision frequency in the source database, Fi (j) is the cumulative decision frequency of the image j for the i-th private classifier in his own area of competence. The voting function qj of the j-th class is represented by the expression: $q_j = \sum_i \mu_{ij}, j = 1,2, \ldots, J$.

If pattern X does not belong to the area of competence of the private i-th classifier, then the value ij = 0. In this case, the results of the work of the private classifier are not taken into account in the future, i.e. the procedure for the reduction of the neural network ensemble is performed. The summation is carried out over all the remaining classifiers. The decision whether pattern X belongs to one of the classes Aj is made in accordance with the rule:

If $q_{j*} = max_j q_j \ , X \in A_{j*}$ Such an approach is not always justified in conditions of noisy input data, and therefore, a refusal from classification may be recorded. The choice of a strategy for combining solutions of private classifiers, as a rule, does not require large computing resources, but at the same time provides a higher quality of a collective solution. In this case, one of the strategies can be used [14]: selection and fusion. In the first case, each subspace of solutions corresponds to a separate classifier, in the second case, particular classifiers are used on the entire space of solutions. Of the technologies that ensure effective projection of solutions of private classifiers onto the target space, the most acceptable are [2, 15]:

- decision templates method (the simplest method);
- weighted averaging;

- the method of multi-tiered generalization (uses a two-stage procedure for generating solutions of classifiers with a nonlinear combination of individual solutions), which has various modifications.

The choice of the preferred aggregation method is based on the rule of minimum classification error.

A computational experiment to test the proposed approach was carried out using test data from the repository [16]. For the experiment, we used two-layer feedforward neural networks. The computational experiment showed an increase in the accuracy of the classification solution based on neural network ensembles by an average of 8–12%.

A promising direction is the development of collective methods for classifying security events, taking into account the significant diversity and correlation of input data. Interesting results can be obtained when using in the architecture of neural network ensembles of neural networks based on new neural-like elements, for example, selective neurons, which are closer to a real biological neuron and have cognitive elements [17, 18]. Selective neural networks do not use weighting factors, which can significantly reduce the amount of computations when training a neural network and increase the efficiency of solving classification problems.

An important direction is the development of classification models based on the application of a multi-agent approach, which allows generating neural networks - ensemble classifiers, taking into account the peculiarities of subject areas and using them as intelligent agents. Of great interest is the development of hybrid classification systems, as there are many alternative systems based on various mathematical models and technologies for a single subject area.

3.2. Algorithms for selecting fuzzy neural networks in an ensemble

Ensemble methods are a machine learning paradigm where multiple models (often referred to as "weak learners") are trained to solve the same problem and combined to produce better results. The main hypothesis is that with the right mix of weak models, we can get more accurate and / or reliable models.

In machine learning, whether we are faced with a classification or regression problem, the choice of model is extremely important to have any chance of getting good results. This choice can depend on many variables of the problem: the amount of data, the dimension of the space, the hypothesis of the distribution ...

The weak bias and variance of the model, although they most often change in opposite directions, are the two most fundamental features expected for a model. Indeed, in order to be able to "solve" a problem, we want our model to have enough degrees of freedom to resolve the underlying complexity of the data we are working with, but we also want it not to have too many degrees of freedom to avoid it. high spread and be more stable. This is a well-known trade-off between offset and spread (fig. 3.2).

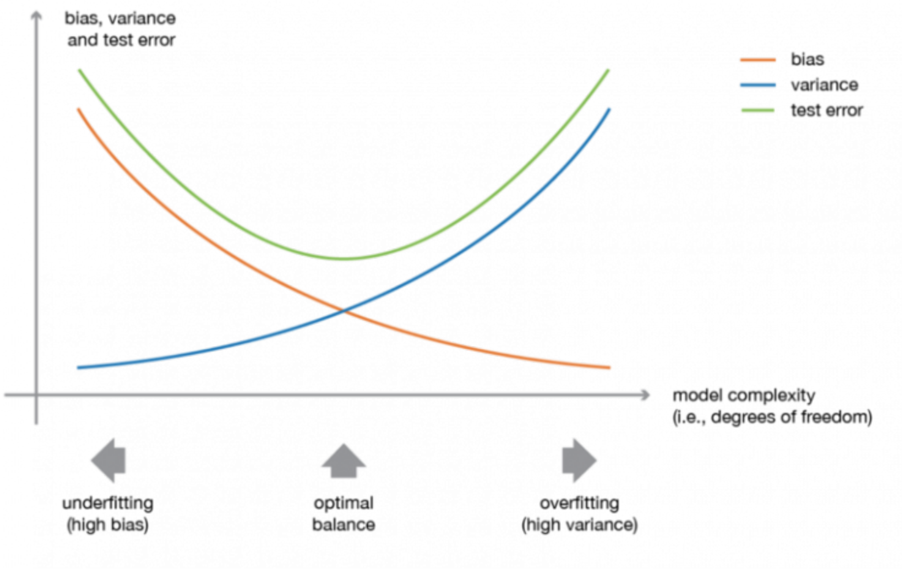


Fig 3.2: An illustration of the trade-off between offset and spread

In ensemble learning theory, we introduce the concepts of weak learners (or base models) that can be used as building blocks for designing more complex models by combining several of them. In most cases, these baseline models do not perform as well on their own due to the fact that they have a high bias (for example, models with a low degree of freedom), or because they have too much spread to be stable (for example, models with a high degree of freedom). Then the idea behind ensemble methods is to try to reduce the bias and / or spread of such weak learners by combining several of them together to create a strong learner (or ensemble model) that achieves better results.

To implement the ensemble method, we first need to select our weak learners to aggregate. Basically (including well-known bagging and boosting methods) there is a single basic learning algorithm, so we have homogeneous weak learners who learn in different ways. The resulting ensemble model is called “homogeneous”. However, there are also some methods that use different types of basic learning algorithms: some dissimilar weak learners are then combined into a "dissimilar ensemble model".

One of the important points is that our selection of weak learners must be consistent with how we aggregate these models. If we select weak learners with low bias but high variance it should be using an aggregation method that tends to reduce the spread, whereas if we select weak learners with low variance but high variance it should be an aggregation method that tends to decrease the offset.

This brings us to the question of how to combine these models. We can mention three main types of meta-algorithms that aim to bring weak learners together:

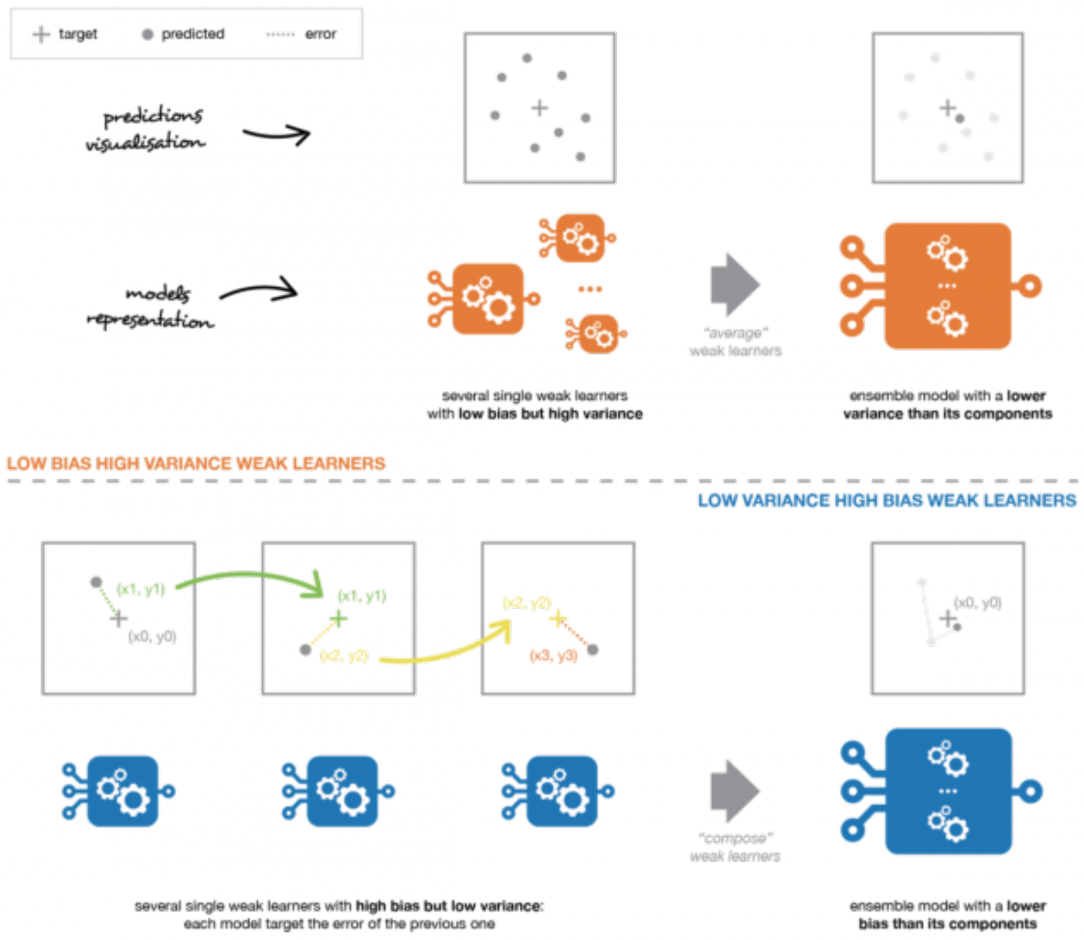
Bagging. In this case, homogeneous weak learners are often considered, trained in parallel and independently of each other, and then combined by following some deterministic averaging process.

Boosting. In this case, homogeneous weak students are often considered, taught in a consistently adaptive way (the weak student depends on the previous ones) and combines them, following a deterministic strategy.

Staking. In this case, dissimilar weak learners are often taken into account, studied in parallel and combined, training a metamodel to derive a prediction based on predictions of various weak models.

Roughly speaking, we can say that bagging will mainly focus on getting an ensemble model with less scatter than its components, while boosting and stacking will mostly try to produce strong models with less offset than their components.

In the following sections, we'll go into more detail about bagging and boosting (which are used a bit more broadly than stacking and allow us to discuss some of the key concepts of ensemble methods) before giving a quick overview of stacking.



target
predicted
error
predictions visualisation
models representation
"average" weak learners
several single weak learners with low bias but high variance
ensemble model with a lower variance than its components
LOW BIAS HIGH VARIANCE WEAK LEARNERS
LOW VARIANCE HIGH BIAS WEAK LEARNERS
(x1, y1)
(x0, y0)
(x2, y2)
(x3, y3)
"compose" weak learners
several single weak learners with high bias but low variance: each model target the error of the previous one
ensemble model with a lower bias than its components

Fig 3.3: Weak learners union

Weak learners (fig. 3.3) can be combined to produce the best performing model. The way in which basic models are combined must be adapted to their types. Low-biased, high-scatter models should be combined to make the strong model more robust, while low-scatter and high-biased models should be combined to make the ensemble model less biased.

In parallel methods, we consider different students independently of each other and thus it is possible to teach them simultaneously. The best known of these is "bootstrap aggregation", which aims to create an ensemble model that is more robust than the individual models that make it up.

Let's start by defining a bootstrap. This statistical method consists of generating samples of size B (the so-called bootstrap samples) from an original dataset of size N by randomly selecting items with repetitions in each case B (fig. 3.2.3).

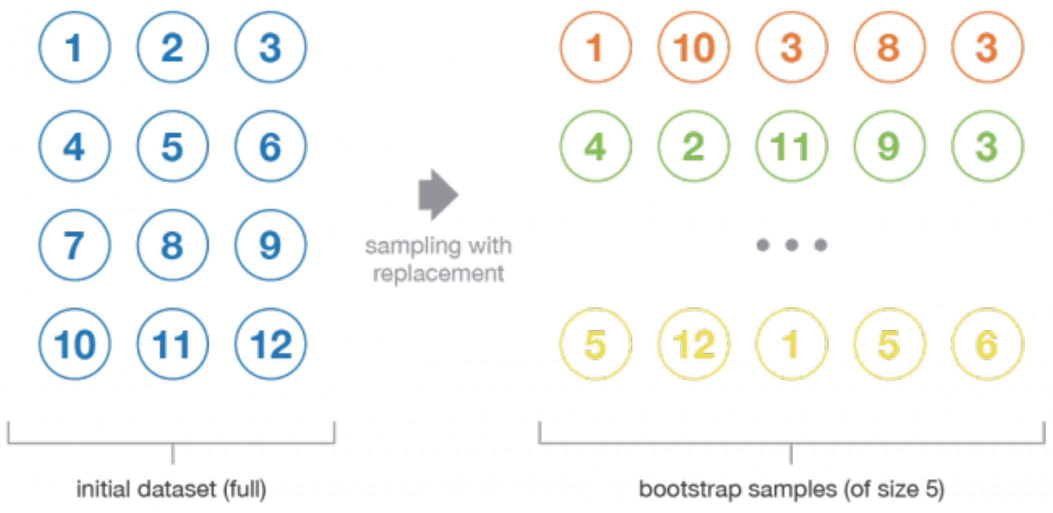


Fig 3.4: Bootstrapping

Under some assumptions, these samples have fairly good statistical properties: in a first approximation, they can be considered either taken directly from the true underlying (and often unknown) data distribution, or independently of each other. Thus, they can be viewed as representative and independent samples of the true distribution of the data (almost identical samples). The hypothesis that must be tested to make this approximation valid has two sides. First, the size N of the original dataset must be large enough to cover most of the complexity of the underlying distribution so that the sample from the dataset is a good approximation to the sample from the real distribution (representativeness). Second, the dataset size N must be large enough compared to the bootstrap sample size B so that the samples do not correlate too much (independence). Please note that in what follows we will sometimes refer to these properties (representativeness and independence) of bootstrap samples: the reader should always remember that this is only an approximation.

Bootstrap samples are often used, for example, to estimate the spread or confidence intervals of statistical estimates. By definition, a statistical estimate is a function of some observations and therefore a random variable with a scatter derived from those observations. To estimate the spread of such an estimate, we need to estimate it on several independent samples taken from the distribution of interest. In most cases, considering truly independent samples would require too much data compared to the amount actually available. We can then use bootstrap to generate multiple bootstrap samples, which can be thought of as “nearly representative” and “nearly independent” (almost “independent equally distributed samples”). These examples of bootstrap samples will allow us to approximate the spread of the estimate by estimating its value for each of them.

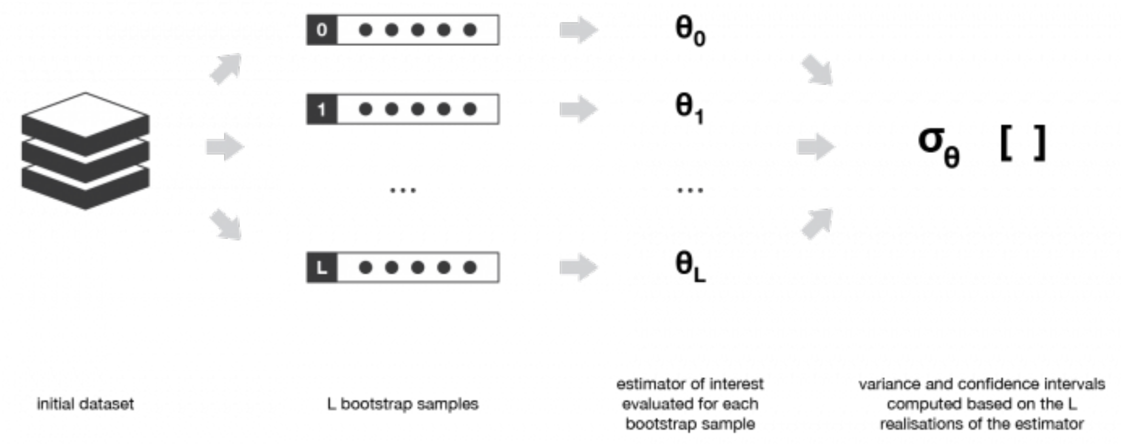

0
1
L
θ0
θ1
θL
σθ [ ]
initial dataset
L bootstrap samples
estimator of interest evaluated for each bootstrap sample
variance and confidence intervals computed based on the L realisations of the estimator

Fig 3.5: Bootstrap scheme

3.2.1. Boosting

In machine learning, **boosting** is an ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones

Boosting methods work in the same spirit as bagging methods: we create a family of models that combine to create a strong learner who performs better. However, unlike bagging, which is mainly aimed at reducing scatter, boosting is a technique that is to adapt sequentially several weak learners in an adaptive way: each model in the sequence is matched, which gives more importance to objects in the dataset that are bad processed by previous models in sequence. Intuitively, each new model focuses its efforts on the most difficult sample objects when training previous models so that we get a strong learner with a lower bias at the end of the process (even if it turns out that boosting will reduce the spread). Boosting, like bagging, can be used for both regression problems and classification.

Basic models that are often considered for boosting are low scatter but high offset models. For example, if we want to use decision trees as our base models, we will mainly choose shallow decision trees with a few nodes deep. Another important reason that motivates the use of low scatter but high bias models as weak learners for boosting is that these models tend to be computationally less expensive (several degrees of freedom in hyperparameter selection). Indeed, since the computation to fit different models cannot be performed in parallel (unlike bagging), it can become too costly to fit multiple complex models sequentially.

Once the weak learners are selected, we still need to determine how they will consistently fit (what information from previous models do we take into account when fitting the current model?) And how they will aggregate (how do we aggregate the current model to the previous ones?). We will discuss these issues in the next two subsections that describe in more detail two important boosting algorithms: ad boost (adaptive boosting) and gradient boosting.

In a nutshell, these two meta-algorithms differ in how they create and integrate weak learners in a sequential process. Adaptive boosting updates the weights attached to each of the objects in the training dataset, while gradient boosting updates the values of those objects. This difference comes from the fact that both methods try to solve the optimization problem, which is to find the best model that can be written as a weighted sum of weak learners.

Focusing on boosting

In sequential methods, various combined weak models are no longer trained independently of each other. The idea is to iteratively train the models in such a way that training the model at this stage depends on the models trained in the previous stages. Boosting is the most famous of these approaches, and it creates an ensemble model that has less bias than its constituent weak learners.

Boosting methods work in the same spirit as bagging methods: we create a family of models that combine to create a strong learner who performs better. However, unlike bagging, which is mainly aimed at reducing scatter, boosting is a technique that is to adapt sequentially several weak learners in an adaptive way: each model in the sequence is matched, which gives more importance to objects in the dataset that are bad processed by previous models in sequence. Intuitively, each new model focuses its efforts on the most difficult sample objects when training previous models so that we get a strong learner with a lower bias at the end of the process (even if it turns out that boosting will reduce the spread). Boosting, like bagging, can be used for both regression problems and classification.

Basic models that are often considered for boosting are low scatter but high offset models. For example, if we want to use decision trees as our base models, we will mainly choose shallow decision trees with a few nodes deep. Another important reason that motivates the use of low scatter but high bias models as weak learners for boosting is that these models tend to be computationally less expensive (several degrees of freedom in hyperparameter selection). Indeed, since the computation to fit different models cannot be performed in parallel (unlike bagging), it can become too costly to fit multiple complex models sequentially.

Once the weak learners are selected, we still need to determine how they will consistently fit (what information from previous models do we take into account when fitting the current model?) And how they will aggregate (how do we aggregate the current model to the previous ones?). We will discuss these issues in the next two subsections that describe in more detail two important boosting algorithms: adaboost (adaptive boosting) and gradient boosting.

In a nutshell, these two meta-algorithms differ in how they create and integrate weak learners in a sequential process. Adaptive boosting updates the weights attached to each of the objects in the training dataset, while gradient boosting updates the values of those objects. This difference comes from the fact that both methods try to solve the optimization problem, which is to find the best model that can be written as a weighted sum of weak learners.

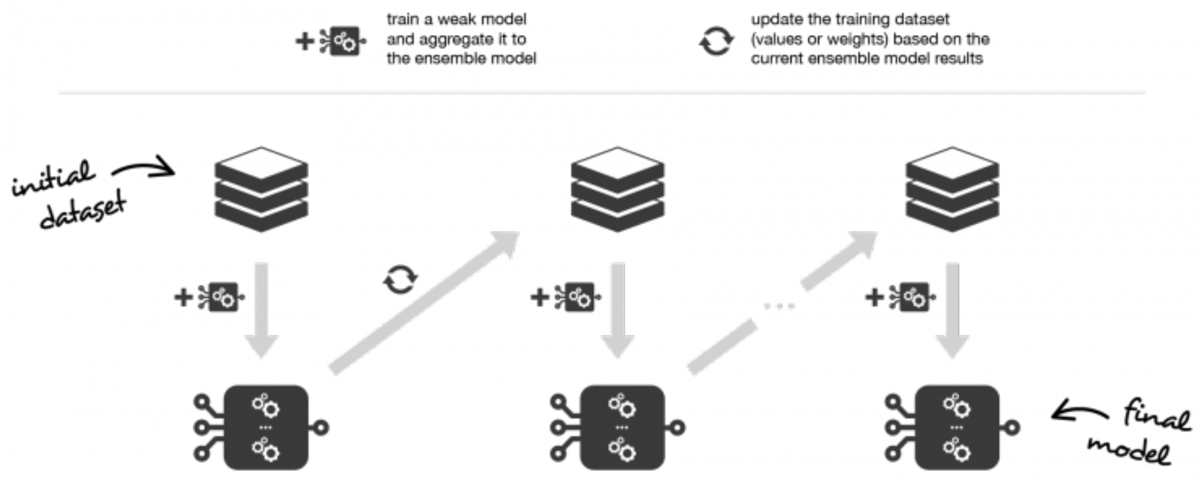

train a weak model and aggregate it to the ensemble model
update the training dataset (values or weights) based on the current ensemble model results
initial dataset
final model

Fig 3.6: Boosting scheme

Boosting consists of iterative selection of a weak student, aggregating it into an ensemble model and "updating" the training dataset to better take into account the strengths and weaknesses of the current ensemble model when selecting the next base model.

Adaptive boosting

With adaptive boosting ("adaboost"), we try to define our ensemble model as a weighted sum of L weak learners $s_L(.) = \sum_{l=1}^{L} c_l * w_l(.)$- where c_l are coefficients and $w_l$are weak learners

Finding the best ensemble model with this ensemble model notation is a challenging optimization task. Instead of trying to solve it analytically in one go (finding all the coefficients and weak learners that give the best overall additive model), we use an iterative optimization process that is much more malleable, even though it may lead to a suboptimal solution. Specifically, we add weak learners one by one, looking through each iteration to find the best possible pair (coefficient, weak learner) to add to the current ensemble model. In other words, we iteratively define (s_l) as $s_l(.) = \ s_{l-1}(.) + c_l * w_l(.)$

where c_l and w_l are chosen so that s_l is the model that best fits the training data, and thus the best possible improvement over s_ (l-1). Then we can define

$$\boldsymbol{(c_l w_l(.)) = \arg min\, E(s_{l-1}(.) + cw(.)) = \arg min \sum_{n=1}^{N} e(y_n, s_{l-1}(x_n) + cw(x_n))}$$

where E (.) is the error of fitting the given model, and e (.,.) is the loss / error function. Thus, instead of global optimization for all L-models in the sum, we approximate the optimum by local optimization by building and adding weak learners to the strong model one at a time.

In particular, when considering binary classification, we can show that the adaboost algorithm can be rewritten into a process that runs as follows. First, it updates the weights of the objects in the dataset and trains a new weak learner, paying particular attention to observations misclassified by the current ensemble model. Second, it adds the weak student to the weighted sum according to the update factor, which expresses the effectiveness of this weak model: the better the weak student does his job, the more it will be counted in the strong student.

So, suppose we are faced with a binary classification problem with N objects in our dataset, and we want to use the adaboost algorithm with a given family of weak models. At the very beginning of the algorithm (the first model of the sequence) all objects have the same weights 1 / N. Then we repeat L times (for L students in the sequence) the following steps:

- train best possible weak model with current observation weights
- calculate the value of the update rate, which is a kind of scalar metric for the weak student's assessment that shows how much the contribution of this weak student should be taken into account in the ensemble model
- upgrade a strong student by adding a new weak student multiplied by his refresh rate
- calculate new object weights that show which observations we would like to focus on in the next iteration (the weights of erroneously predicted objects increase in the aggregate model, and the weights of correctly predicted objects decrease)

By repeating these steps, we then sequentially build our L models and combine them into a simple linear combination, weighted by coefficients expressing the performance of each student. Note that there are variations on the original adaboost algorithm such as LogitBoost (classification) or L2Boost (regression), which mainly differ in their choice of loss function.

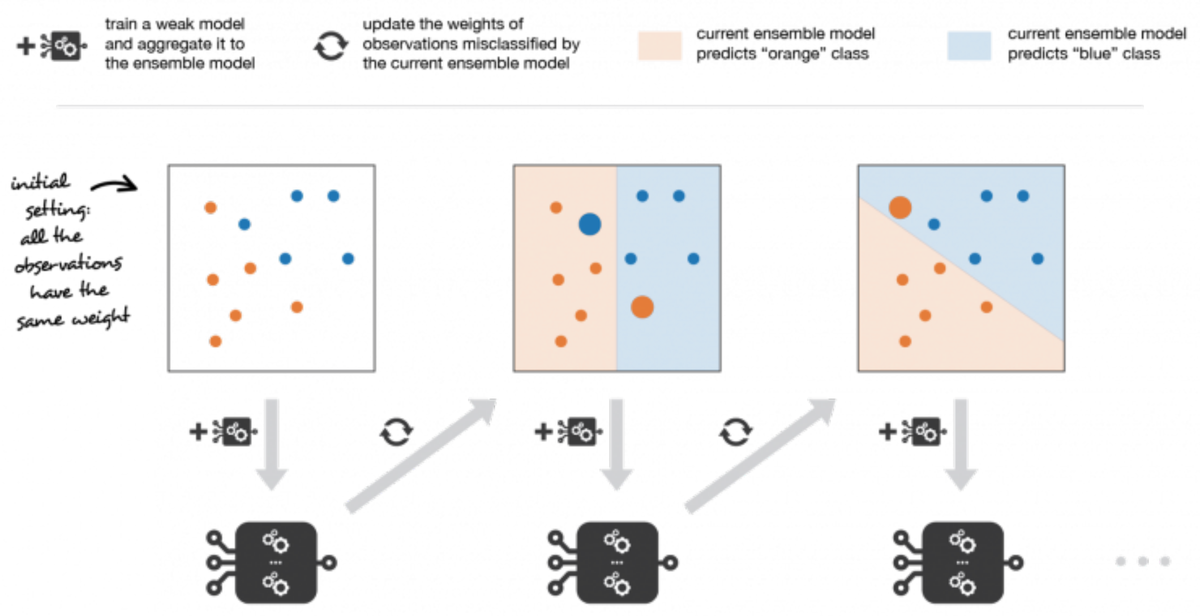

train a weak model and aggregate it to the ensemble model
update the weights of observations misclassified by the current ensemble model
current ensemble model predicts "orange" class
current ensemble model predicts "blue" class
initial setting: all the observations have the same weight

Fig 3.7: Adaboost

Adaboost (fig 3.7) updates the object weights on each iteration. The weights of well-classified objects are reduced relative to the weights of misclassified objects. The models that perform better carry more weight in the final ensemble model.

Gradient boosting

With gradient boosting, the ensemble model we are trying to build is also a weighted sum of weak learners $s_L(.) = \sum_{l=1}^{L} c_l * w_l(.)$- where $c_l$are coefficients and $w_l$are weak learners.

As we mentioned for adaboost, it is too difficult to find the optimal model in this form of notation of the ensemble model, and an iterative approach is required. The main difference from adaptive boosting is the definition of a sequential optimization process. Who would have thought, gradient boosting reduces the problem to gradient descent: at each iteration, we fit the weak student to the anti gradient of the current fitting error with respect to the current ensemble model.

Let's try to clarify this last point. First, the theoretical ensemble model gradient descent process can be written as $s_l(.) = s_{l-1}(.) - c_l * \nabla_{s_{l-1}} E(s_{l-1})(.)$

where E (.) is the error of fitting the given model, c_l is the coefficient corresponding to the step size, and$-\nabla_{s_{l-1}} E(s_l - 1)(.)$ is the anti gradient of the fitting error relative to the ensemble model at step l_1. This (rather abstract) anti gradient is a function that in practice can only be estimated for objects in the training set (for which we know the inputs and outputs): these estimates are called pseudo-residuals attached to each object. Moreover, even if we know the observational values of these pseudo-residuals, we do not want to add any function to our ensemble model: we only want to add a new instance of the weak model. So the natural thing to do is teach the weak learner pseudo-residuals for each observation. Finally, the coefficient c_l is computed according to a one-dimensional optimization process (linear search to obtain the best step size c_l).

So, suppose we want to use gradient boosting with a family of weak models. At the very beginning of the algorithm (the first model of the sequence), pseudo-residuals are set equal to the values of the objects. Then we repeat L times (for L sequence models) the following steps:

- Teach the best possible weak student with pseudo-residuals (bring the anti gradient closer to the current strong student)
- calculate the value of the optimal step size, which determines how much we update the ensemble model towards the new weak student
- update the ensemble model by adding a new weak learner multiplied by the step size (do a gradient descent step)
- compute new pseudo-residuals that indicate for each observation in which direction we would like to update the following ensemble model predictions

By repeating these steps, we sequentially build our L models and aggregate them according to a gradient descent approach. Note that although adaptive boosting tries to solve the “local” optimization problem at each iteration (find the best weak learner and its coefficient to add to the strong model), gradient boosting uses a gradient descent approach instead and is easier to adapt to a large number of loss functions. Thus, gradient boosting can be viewed as a generalization of adaboost for arbitrary differentiable loss functions.

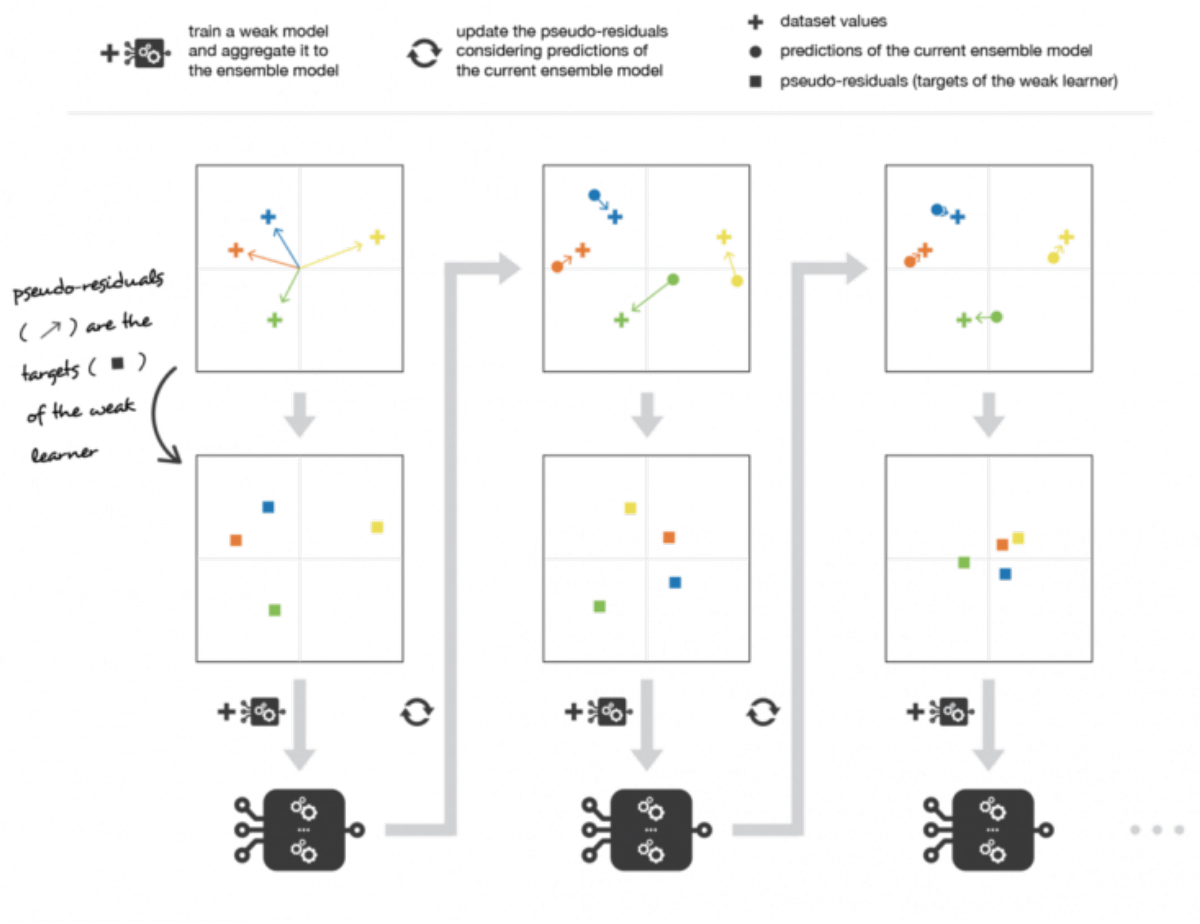

train a weak model and aggregate it to the ensemble model
update the pseudo-residuals considering predictions of the current ensemble model
dataset values
predictions of the current ensemble model
pseudo-residuals (targets of the weak learner)
pseudo-residuals (↗) are the targets (■) of the weak learner

Fig 3.8: Gradient boosting

Gradient boosting updates the observation values at each iteration. Weak learners are trained to fit pseudo-residuals, which indicate in which direction to adjust the predictions of the current ensemble model to reduce error.

3.2.2. Bagging

Bootstrap aggregating, also called **bagging** (from bootstrap aggregating), is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting.

When training a model, regardless of whether we are dealing with a classification or regression problem, we get a function that takes input, returns an output, and is defined against the training dataset. Due to the theoretical spread of the training dataset (we recall that the dataset is an observable sample coming from a truly unknown underlying distribution), the fitted model is also subject to variability: if a different dataset were observed, we would get a different model.

The idea of bagging in this case is simple: we want to pick up several independent models and “average” their predictions in order to get a model with less scatter. In practice, however, we cannot find completely independent models, because this would require too much data. Thus, we rely on the good "roughness" of bootstrap samples (representativeness and independence) to find models that are nearly independent.

First, we generate several bootstrap samples so that each new bootstrap sample acts as (almost) another independent dataset taken from the true distribution. Then we can train a weak learner for each of these samples and, finally, aggregate them so that we sort of "average" their results and, thus, get an ensemble model with a scatter less than its individual components. Roughly speaking, since bootstrap samples are roughly independent and equally distributed, the same is true for trained weak learners. Then "averaging" the results of weak students does not change the expected answer, but decreases its spread (just as averaging of independent, equally distributed random variables retains the expected value, but decreases the spread).

So, suppose we have L bootstrap samples (approximations of L independent datasets) of size B. This is denoted:

$$\{z_1^1, z_2^1, ..., z_B^1\}, \{z_1^2, z_2^2, ..., z_B^2\}, ..., \{z_1^L, z_2^L, ..., z_B^L\} \qquad z_b^l \equiv b\text{-th observation of the } l\text{-th bootstrap sample}$$

We can train L almost independent weak learners (one for each dataset):

$$w_1(.), w_2(.), ..., w_L(.)$$

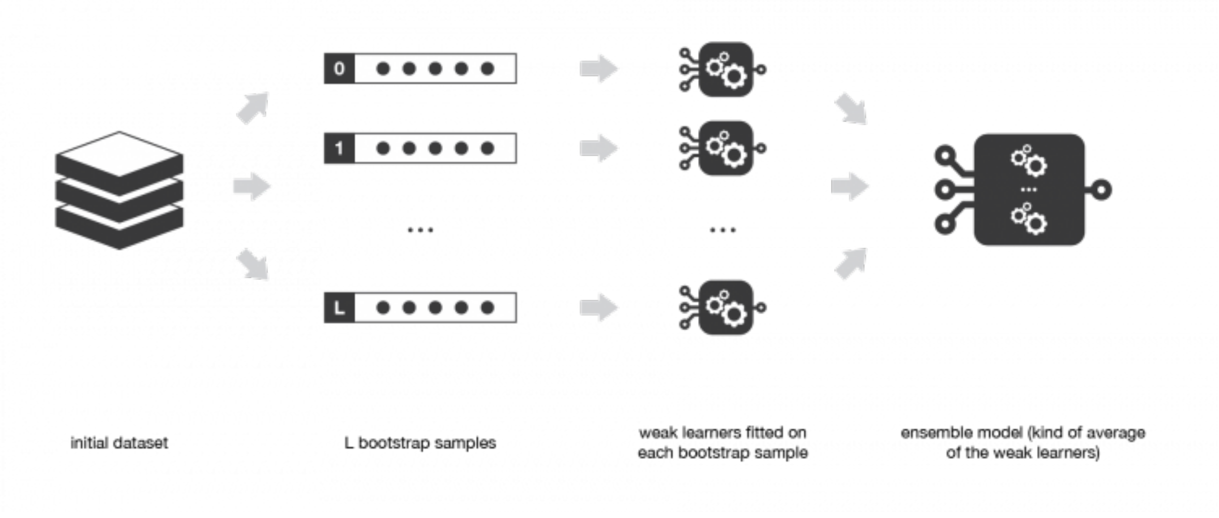
And then we combine them with some averaging process to get an ensemble model with less scatter. For example, we can define our strong model so that

$s_L(.) = \frac{1}{L}\sum_{l=1}^{L} w_l(.)$- simple average, for regression problem

$s_L(.) = \arg\max[card(l|w_l(\llbracket.) = k)]\rrbracket$ - simple majority vote, for classification problem

There are several possible ways to combine multiple models trained in parallel. For a regression problem, the outputs of the individual models can literally be averaged to produce the output of the ensemble model. For a classification problem, the class predicted by each model can be thought of as a vote, and the class that receives the most votes is the response of the ensemble model (this is called majority voting). For the classification problem, we can also consider the probabilities of each class predicted by all models, average those probabilities, and keep the class with the highest average probability (this is called soft voting). Averages or votes can be simple or weighted if any appropriate weights are used.

Finally, we can mention that one of the big advantages of bagging is its concurrency. Since the different models are trained independently of each other, intensive parallelization techniques can be used if necessary.



0
1
...
L
initial dataset
L bootstrap samples
weak learners fitted on each bootstrap sample
ensemble model (kind of average of the weak learners)

Fig 3.9: Bagging

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees

Decision trees are very popular base models for ensemble methods. Strong learners made up of multiple decision trees can be called "forests." The trees that make up the forest can be chosen to be either shallow (a few nodes deep) or deep (many nodes deep, if not full depth with all leaves). Shallow trees have less spread, but higher offset, and then sequential methods will be the best choice for them, which we will describe later. Deep trees, on the other hand, have low offset but high spread and are thus a good choice for bagging that is mainly aimed at reducing spread.

Random forest is a bagging technique where deep trees trained on bootstrap samples are combined to produce a lower scatter result. However, random forests also use a different trick to make several trained trees less correlated with each other: when constructing each tree, instead of selecting all features from the dataset to generate a bootstrap, we select and store only a random subset of them to build the tree (usually the same for all bootstrap samples).

Feature sampling does result in all trees not looking at the same information to make their decisions and thus decreasing the correlation between different returned outputs. Another advantage of feature sampling is that it makes the decision-making process more robust to missing data: observation values (from the training dataset or not) with missing data can be reconstructed using regression or tree-based classification that only consider features. where data is not missing. Thus, the random forest algorithm combines the concepts of bagging and random object subspace selection to create more robust models.

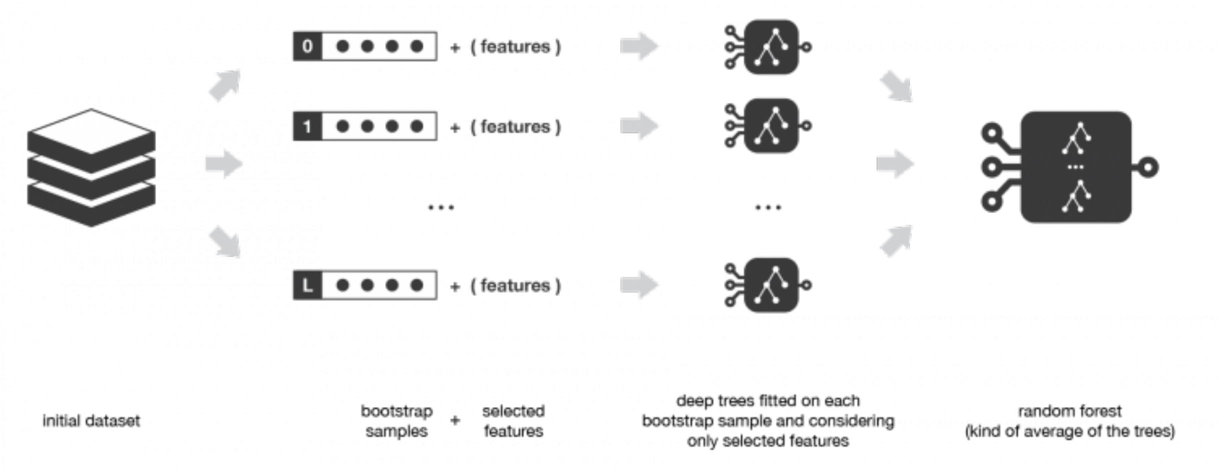

0
+ (features)
1
+ (features)
...
L
+ (features)
initial dataset
bootstrap samples
+
selected features
deep trees fitted on each bootstrap sample and considering only selected features
random forest
(kind of average of the trees)

Fig 3.10: Random forest method

The random forest method is a bagging method with decision trees as weak learners. Each tree is placed in a bootstrap sample, only taking into account a random selection of a subset of features.

Focusing on boosting

In sequential methods, various combined weak models are no longer trained independently of each other. The idea is to iteratively train the models in such a way that training the model at this stage depends on the models trained in the previous stages. Boosting is the most famous of these approaches, and it creates an ensemble model that has less bias than its constituent weak learners.

3.2.3. Staking

Staking has two main differences from bagging and boosting. Firstly, stacking often takes into account dissimilar weak learners (different learning algorithms are combined), while bagging and boosting take into account mostly homogeneous weak learners. Second, stacking teaches how to combine basic models using a metamodel, while bagging and boosting teaches weak learners to combine using deterministic algorithms.

As we mentioned, the idea of stacking is to train several different weak learners and combine them by training a metamodel to infer predictions based on the multiple predictions returned by these weak models. So we need to define two things to build our stack model: the L students we want to train and the metamodel that brings them together.

For example, for a classification problem, we can choose the KNN classifier, logistic regression and SVM as a weak learner and decide to train the neural network as a metamodel. The neural network will then take the results of our three weak students as input and learn to make final predictions based on them.

So, suppose we want to train a stacked ensemble of L weak learners. Then we have to follow these steps:

- split training data in two
- select L weak learners and train them on the data of the first fold (part)
- for each of the L weak learners, make predictions for the objects from the second fold
- train the metamodel a second time using predictions made by weak learners as input

In the previous steps, we split the dataset into two parts, because the data predictions that were used to train weak learners have nothing to do with training the metamodel. Thus, the obvious disadvantage of this dividing our dataset into two parts is that we only have half of the data for training the base models and half of the data for training the metamodel. To overcome this limitation, we can, however, follow some kind of "k-fold cross-learning" approach (similar to what is done in k-fold cross-validation), so all objects can be used to train the meta-model: for of any object, the prediction of weak learners is done using examples of these weak learners trained on k-1 folds that do not contain the object in question. In other words, he learns k-1 folds to make predictions about the remaining fold for objects in any folds. Thus, we can create appropriate predictions for each object in our dataset, and then train our metamodel on all of these predictions.

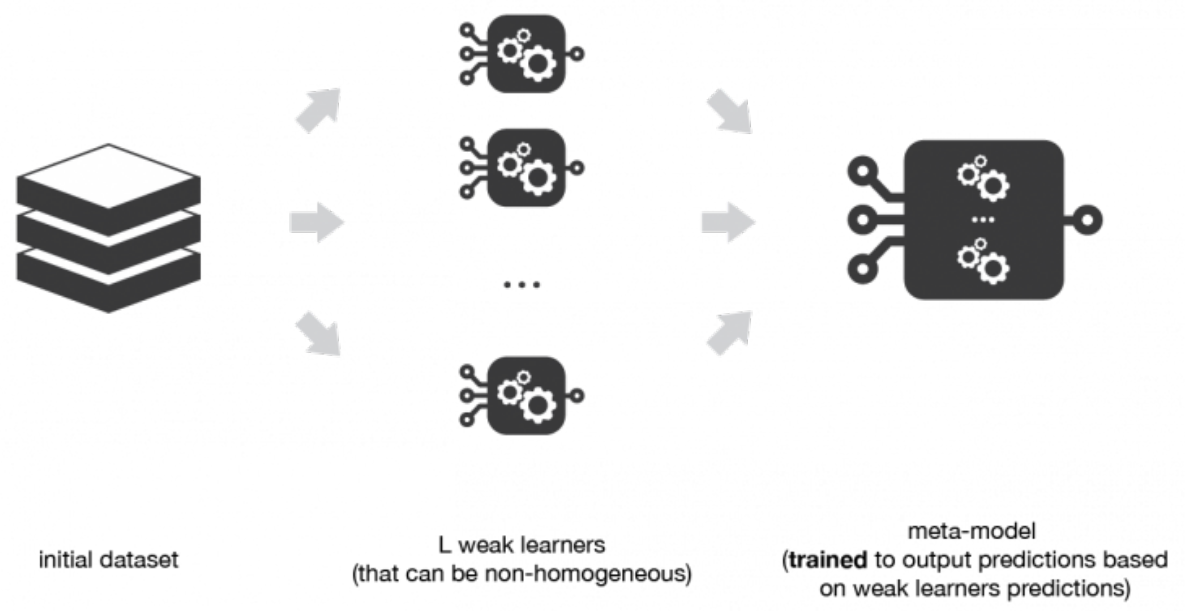

initial dataset
L weak learners
(that can be non-homogeneous)
meta-model
(**trained** to output predictions based on weak learners predictions)

Fig 3.11: Staking representation

Stacking consists of training a metamodel to produce results based on the results obtained by a few weak lower-level students

Multilevel stacking

A possible extension of staking is tiered stacking. It consists of stacking with multiple layers. As an example, let's consider 3-level stacking. At the first level (layer) we come to the L weak learners that have been selected. Then, at the second level, instead of training one metamodel to predict weak models (as described in the previous subsection), we train M such metamodels. Finally, at the third level, we train the last metamodel, which takes as input the predictions returned by the M metamodels of the previous level.

From a practical point of view, note that for each metamodel of different levels of the stacked ensemble layered model, we have to choose a learning algorithm, which can be almost anything we want (even algorithms already used at lower levels) .We can also mention. that adding layers can be data-demanding (if a k-fold-like technique is not used, then more data is required) or time-consuming (if a k-fold-like technique is used and many models have to be trained).

Conclusions: so, which is better: Bagging or Boosting? There is no clear winner; it depends on the data, modeling and circumstances. Enhance and increase the variance of your single score as they combine multiple scores from different models. So that as a result can get a model with higher stability.

If the problem is that one model has very low performance, then the displacement in the bags will be better. What's less, Boosting can create a combined model with fewer errors, by optimizing the benefits and reducing the pitfalls of a single model.

On the contrary, if the complexity of single models is due to them, the best option would be bagging. Reinforcement, in turn, does not help to avoid retraining; In fact, this technique itself faces this problem. For this reason, bagging is more effective than busting.

CHAPTER 4
SOFTWARE FOR SOLVING THE PROBLEM OF STRUCTURAL-PARAMETRIC SYNTHESIS

4.1. Software description

In order to study the artificial neural network (ANN) and predicting using ANN, it was decided to use one of the popular frameworks instead of creating an independent module (due to the complexity of debugging and the complexity of implementing GPU software), Table 4.1 lists the popular today's frameworks for the study of artificial neural networks.

In order to finish the task in this work, it was decided to make the most of the hardware capabilities of modern computers, in particular, the use of NVIDIA GPU. This makes it possible to accelerate the execution of matrix operations (which are the main operations in the training of artificial neural networks) in 10-20 times compared with the performance of the Intel CPU and integrated GPU. This allows you to quickly teach the chosen architecture of the neural network and, based on the results obtained, quickly change it.

Table 4.1

The most popular libraries for the creation of ANN

Name	Manufacturer	Open source	Platform	Language	Interface	CUDA	DNN	Parallel work
Apache Singa	Apache Incubator	+	Unix, Windows	C++	Python, C++, Java	+	+	+
Caffe	Berley Learning Center	+	Unix, Windows	C++	Python, MATLAB	+	+	-
Deeplearning4j	Skymind	+	Unix, Windows	Java	Java, Python	+	+	+
Dlib	Devis King	+	Unix, Windows	C++	C++	+	-	+
Keras	Fransua Sholle	+	Unix, Windows	Python	Python	+	+	+
Microsoft Cognative Toolkit	Microsoft	+	Unix, Windows	C++	Python, C++	+	+	+
MXNet	Machine Learning Community	+	Unix, Windows	C++	C++, Python	+	+	+

Neural Designer	Artelnics	-	Unix, Windows	C++	PC	-	-	-
Open NN	Artelnics	+	Unix, Windows	C++	C++	-	-	-
TensorFlow	Google Brain	+	Unix, Windows	C++, Python	Python, C/C++, Go	+	+	+
Theano	Monreal	+	Unix, Windows	Python	Python	+	+	+
Torch	Ronan Collober	+	Unix, Windows	C, Lua	Lua, C	+	+	+
Mathematica	Wolfram Research	+	Unix, Windows	C++	Java, C++	+	+	+

Thus, the list narrowed to Torch, Theano, TensorFlow, MXNet, Keras, Microsoft's Sognitive Toolkit. The choice in this work fell into the open neural network Keras library, because it is constantly improving, stable and able to work on different platforms. It is written in Python's programming language by the Google engineer François Scholl. The Keras library is an add-on for the Deeplearning4j, TensorFlow and Theano frameworks. These frameworks and the Keras library use open source BSD, MIT, Apache 2.0.

Open Source is a license that allows any person to be freely used or distributed commercially or non-commercially, and allows the creation of derivative works (translations, adaptations, remixes, just modified versions, etc.) and distribute such works on the same terms. These licenses are very similar and they all have permission licenses, that is, they allow programmers to use the licensed code in the closed software, provided that the license text is provided with this software.

During work the author used the Python programming language, because Python is best suited for such tasks because it is fairly self-explanatory compared to other languages. Moreover, it has excellent data processing performance.

Large selection of libraries and frameworks

One of the main reasons why Python is used for machine learning is because it has many frameworks that simplify the coding process and reduce development time.

Let's discuss exactly which Python libraries and frameworks are used in machine learning. Scientific computing uses Numpy, advanced computing uses SciPy, and data mining and analysis uses SciKit-Learn. These libraries work in frameworks such as TensorFlow, CNTK, and Apache Spark.

There is a Python framework designed specifically for machine learning - PyTorch.

Intelligibility

Python is the most high-level and easy-to-understand language to work with. Its conciseness and ease of reading make it well suited for teaching software development.

Also, Python is well suited for machine learning because the machine learning algorithms themselves are difficult to understand. When working with Python, the developer does not need to pay much attention to directly writing the code: he can focus all his attention on solving more complex problems related to machine learning.

Python's simple syntax helps the developer test complex algorithms with a minimum of time to implement.

Extensive support

Another advantage of Python is its extensive support and quality documentation. There are many useful resources about Python that a programmer can get help and advice from at any stage of development.

Flexibility

Another advantage of Python in machine learning is its flexibility: for example, a developer has a choice between an object-oriented approach and scripting. Python helps to combine different types of data. Moreover, Python is especially convenient for those developers who write most of their code using an IDE.

Popularity

As noted, Python has gained popularity due to its simple and straightforward syntax structure. This is why there are many Python developers on the market who are willing to work on machine learning related projects.

PyTorch is an open source machine learning framework for Python built on top of Torch. It is used to solve various problems: computer vision, natural language processing. Developed primarily by the Facebook artificial intelligence group. Also, an ecosystem is built around this framework, consisting of various libraries developed by third-party teams: Fast.ai, which simplifies the process of training models, Pyro, a module for probabilistic programming, from Uber, Flair, for natural language processing and Catalyst, for training DL and RL models.

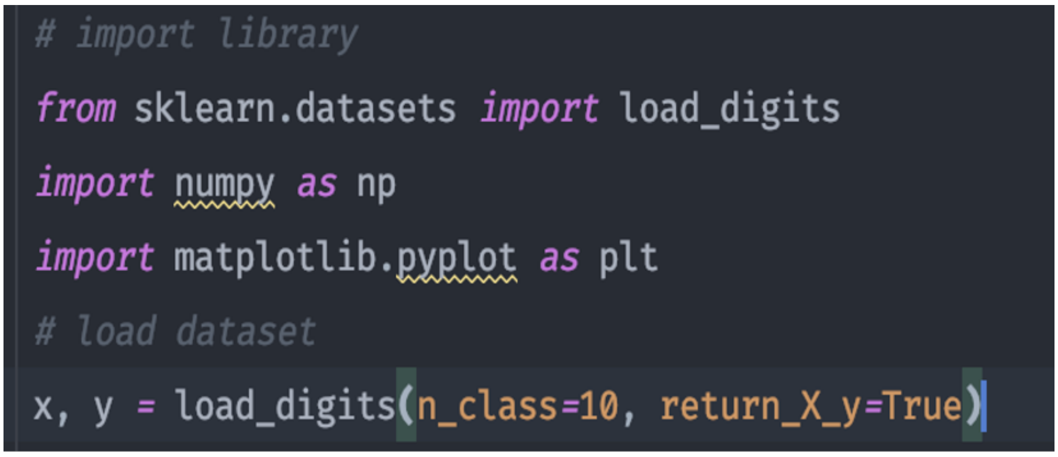
PyTorch provides two main high-level models:

- Tensor computation (similar to NumPy) with advanced GPU acceleration support
- Deep neural networks based on autodiff system

Scikit-learn (also known as sklearn or scikits.learn) is a free machine learning software library for the Python programming language that provides functionality for creating and training various classification, regression, and clustering algorithms, such as linear regression, random forest, gradient boosting. , and works in conjunction with the NumPy and SciPy libraries. Scikit-learn is one of the most popular machine learning libraries.

4.2. Input data description

So, author of this work will work with the Digits dataset from Sklearn, since it can be quite simply obtained in two lines of code (fig. 4.1)



```
# import library
from sklearn.datasets import load_digits
import numpy as np
import matplotlib.pyplot as plt
# load dataset
x, y = load_digits(n_class=10, return_X_y=True)
```

Fig 4.1: Library initialization

Let's see what our data looks like (fig. 4.2):

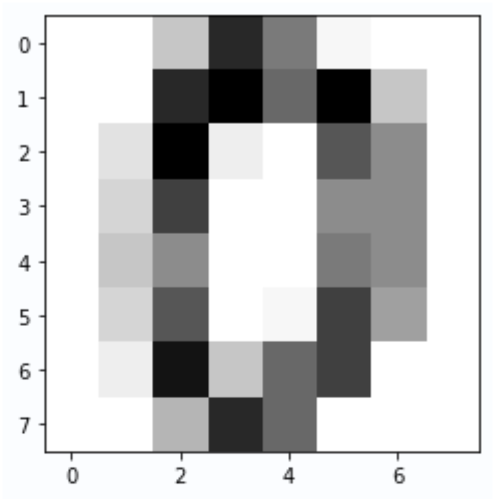
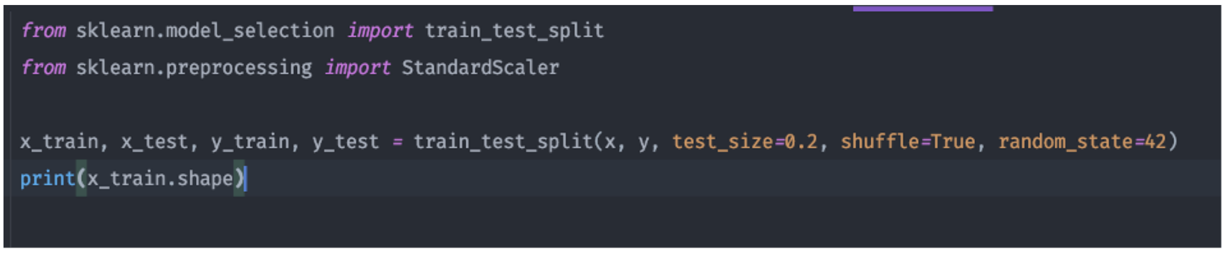


Fig 4.2 Dataset element example

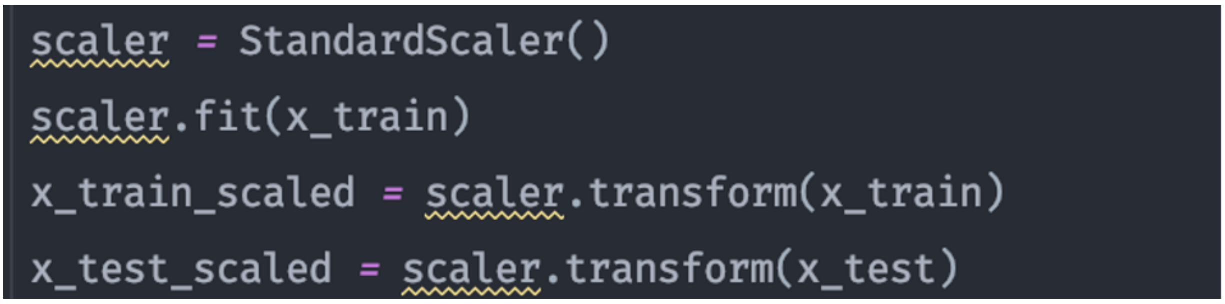
Split our sample into train and test examples fig 4.3



```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, shuffle=True, random_state=42)
print(x_train.shape)
```

Fig 4.3: Split sample

Author use StandardScaler: it converts all data into the range from -1 to 1. We train it only on the training sample so that it does not adjust to the test - so the score of the model after the test will be closer to the score on real data (fig 4.4):



```
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Fig 4.4: Convert data range [-1; 1]

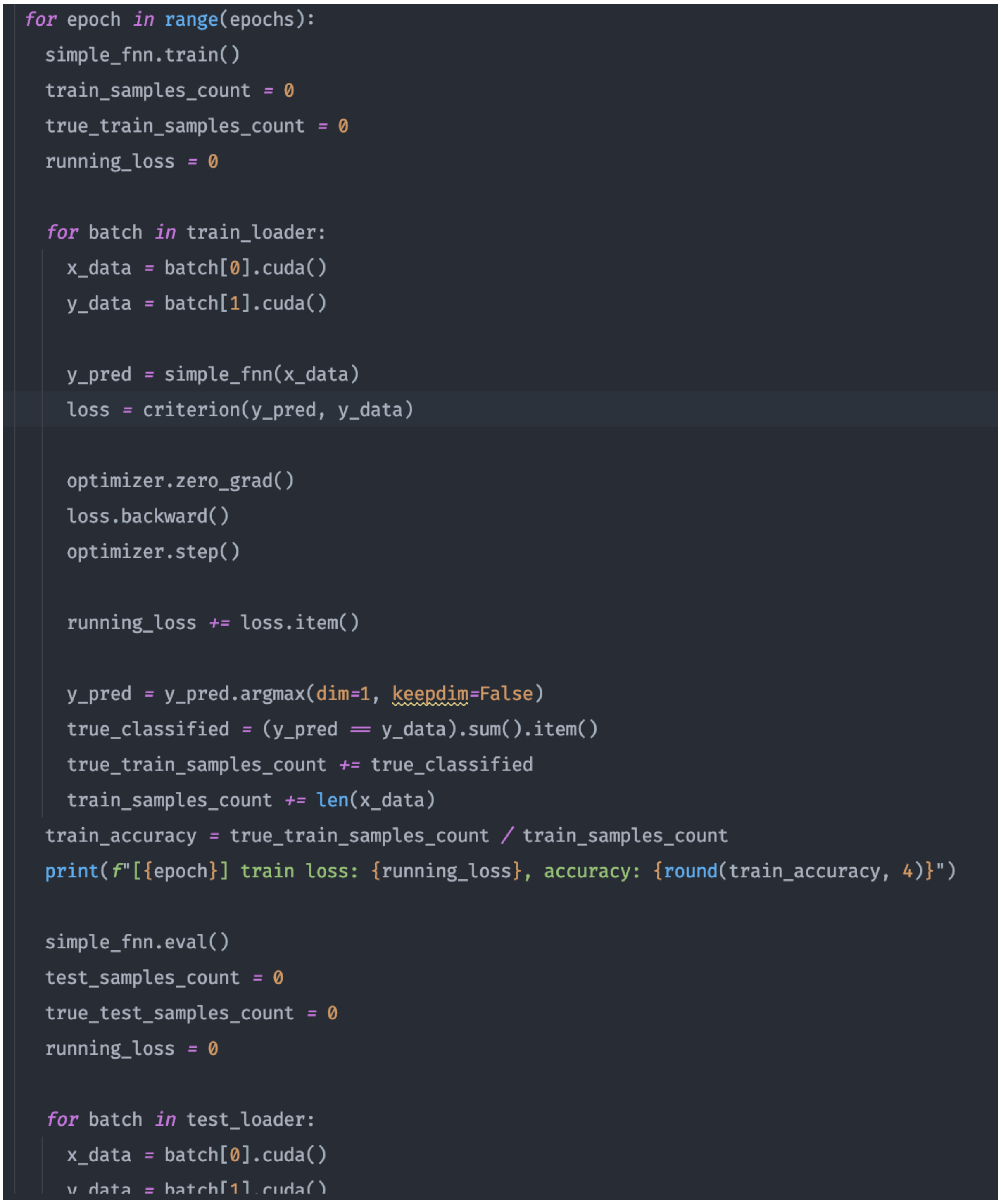
Author will use set of testing and training data in next chapter, with such set we easily can learn our network.

4.3. Solving an applied problem using a neural network

Learn process displayed on screen Fig 4.5, learning algorithm can be described in such way:

- create our network from its constructor and parameters
- `** self.net_params` means that the key-value pairs from the dictionary will be passed to the function as the names of the arguments and their values
- move the network to gpu if we learn on it
- like the network, we create an optimizer
- the first parameter to it is to pass the trained parameters of the neural network
- looking for unique classes
- one of the ensemble requirements in sklearn is the presence of the `classes_` variable in the base model
- change the size of the input data to the dimension of the input layer of the neuron
- similar to the first neural network, create a dataset and DataLoader
- run the data through the model
- backpropagation of the error
- looking for the index of the maximum probability, this is the predicted class
- looking for the number of correctly predicted objects
- count accuracy

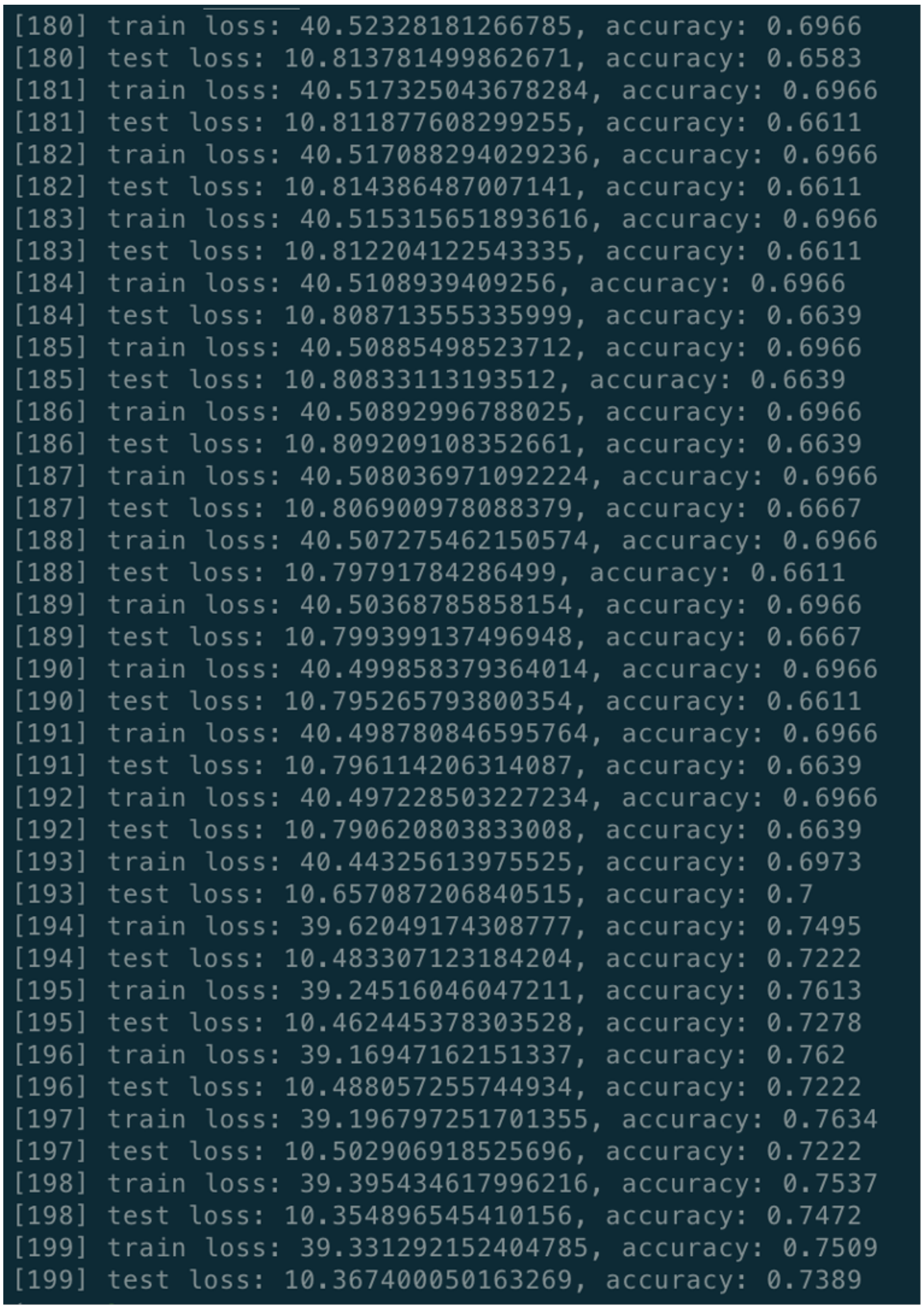
- see what is the maximum difference in accuracy for the last `tol_epochs` epochs
- if the difference is small, stop learning



```
for epoch in range(epochs):
  simple_fnn.train()
  train_samples_count = 0
  true_train_samples_count = 0
  running_loss = 0

  for batch in train_loader:
    x_data = batch[0].cuda()
    y_data = batch[1].cuda()

    y_pred = simple_fnn(x_data)
    loss = criterion(y_pred, y_data)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    running_loss += loss.item()

    y_pred = y_pred.argmax(dim=1, keepdim=False)
    true_classified = (y_pred == y_data).sum().item()
    true_train_samples_count += true_classified
    train_samples_count += len(x_data)
  train_accuracy = true_train_samples_count / train_samples_count
  print(f"[{epoch}] train loss: {running_loss}, accuracy: {round(train_accuracy, 4)}")

  simple_fnn.eval()
  test_samples_count = 0
  true_test_samples_count = 0
  running_loss = 0

  for batch in test_loader:
    x_data = batch[0].cuda()
    y_data = batch[1].cuda()
```

Fig 4.5: Training algorithm

Due to some features of Torch and algorithms, the result on your machine may not match the one obtained here. It's quite normal. Lest consider last 20 epochs



```
[180] train loss: 40.52328181266785, accuracy: 0.6966
[180] test loss: 10.813781499862671, accuracy: 0.6583
[181] train loss: 40.517325043678284, accuracy: 0.6966
[181] test loss: 10.811877608299255, accuracy: 0.6611
[182] train loss: 40.517088294029236, accuracy: 0.6966
[182] test loss: 10.814386487007141, accuracy: 0.6611
[183] train loss: 40.515315651893616, accuracy: 0.6966
[183] test loss: 10.812204122543335, accuracy: 0.6611
[184] train loss: 40.5108939409256, accuracy: 0.6966
[184] test loss: 10.808713555335999, accuracy: 0.6639
[185] train loss: 40.50885498523712, accuracy: 0.6966
[185] test loss: 10.80833113193512, accuracy: 0.6639
[186] train loss: 40.50892996788025, accuracy: 0.6966
[186] test loss: 10.809209108352661, accuracy: 0.6639
[187] train loss: 40.508036971092224, accuracy: 0.6966
[187] test loss: 10.806900978088379, accuracy: 0.6667
[188] train loss: 40.507275462150574, accuracy: 0.6966
[188] test loss: 10.79791784286499, accuracy: 0.6611
[189] train loss: 40.50368785858154, accuracy: 0.6966
[189] test loss: 10.799399137496948, accuracy: 0.6667
[190] train loss: 40.499858379364014, accuracy: 0.6966
[190] test loss: 10.795265793800354, accuracy: 0.6611
[191] train loss: 40.498780846595764, accuracy: 0.6966
[191] test loss: 10.796114206314087, accuracy: 0.6639
[192] train loss: 40.497228503227234, accuracy: 0.6966
[192] test loss: 10.790620803833008, accuracy: 0.6639
[193] train loss: 40.44325613975525, accuracy: 0.6973
[193] test loss: 10.657087206840515, accuracy: 0.7
[194] train loss: 39.62049174308777, accuracy: 0.7495
[194] test loss: 10.483307123184204, accuracy: 0.7222
[195] train loss: 39.24516046047211, accuracy: 0.7613
[195] test loss: 10.462445378303528, accuracy: 0.7278
[196] train loss: 39.16947162151337, accuracy: 0.762
[196] test loss: 10.488057255744934, accuracy: 0.7222
[197] train loss: 39.196797251701355, accuracy: 0.7634
[197] test loss: 10.502906918525696, accuracy: 0.7222
[198] train loss: 39.395434617996216, accuracy: 0.7537
[198] test loss: 10.354896545410156, accuracy: 0.7472
[199] train loss: 39.331292152404785, accuracy: 0.7509
[199] test loss: 10.367400050163269, accuracy: 0.7389
```

Fig 4.6: Train result

We can see that the model has been retrained: the quality on the training sample is higher than the quality on the test sample. So, what do we have: only 74% of the predictions turn out to be correct.

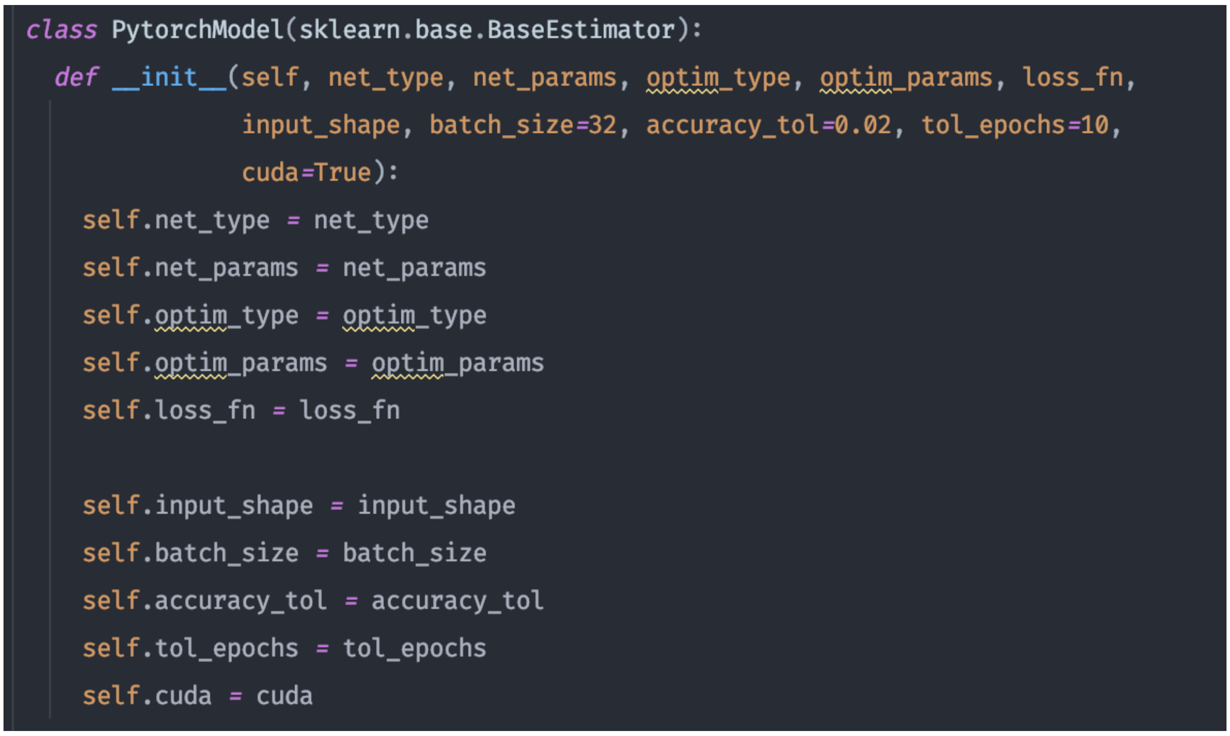
Let’s use the bagging algorithm, Sklearn has the ensembles module, which provides several algorithms for implementing ensembles. Our case is classification, we use BaggingClassifier from sklearn.ensemble

Ensembles operate with basic models. In the case of sklearn, the base model is the entity that implements the sklearn.base.BaseEstimator methods (fig 4.7). Depending on the type of ensemble, different implementations of the functions of this entity will be needed. For our case, it is necessary to describe the fit method (for training the model), the predict_proba function, which gives the probability of each class (so that the metamodel can estimate the confidence of the base classifiers in probabilities and assign their votes higher or lower weights), and the predict function (which gives the number class) so that it is convenient to evaluate the quality of the model.

Base model class constructor. Of those rakes that we managed to stumble upon, it is worth noting the naming of parameters and their types.

Firstly, if you do not want to rewrite the cloning functions of the base classifier object, then you need to give the arguments in the constructor the same names as the assigned class variables. That is, if you supply the netType parameter to the constructor, then in the __init__ method you must declare the netType variable.

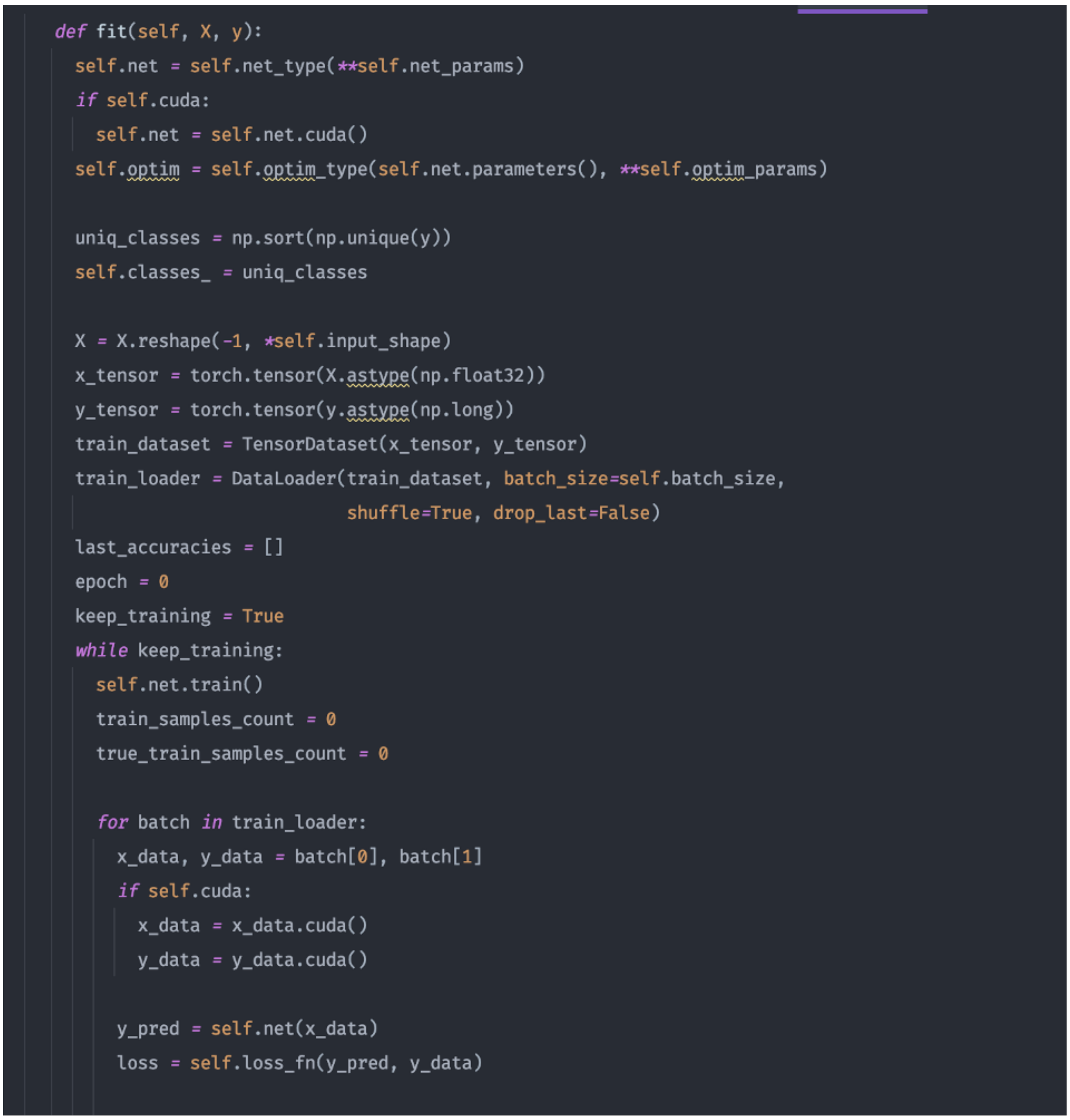
Secondly, it is better to pass as arguments to the constructor not objects, but functions for their creation, because when copying base models (and sklearn copies the model you passed several times until it receives the required number of base models) various difficulties arise with copy functions, deepcopy and object references. For example, it may turn out that the passed and copied optimizer object does not refer to the new (copied) model object, but to the very first one used when initializing the ensemble.



```
class PytorchModel(sklearn.base.BaseEstimator):
  def __init__(self, net_type, net_params, optim_type, optim_params, loss_fn,
               input_shape, batch_size=32, accuracy_tol=0.02, tol_epochs=10,
               cuda=True):
    self.net_type = net_type
    self.net_params = net_params
    self.optim_type = optim_type
    self.optim_params = optim_params
    self.loss_fn = loss_fn

    self.input_shape = input_shape
    self.batch_size = batch_size
    self.accuracy_tol = accuracy_tol
    self.tol_epochs = tol_epochs
    self.cuda = cuda
```

Fig 4.7: Model description

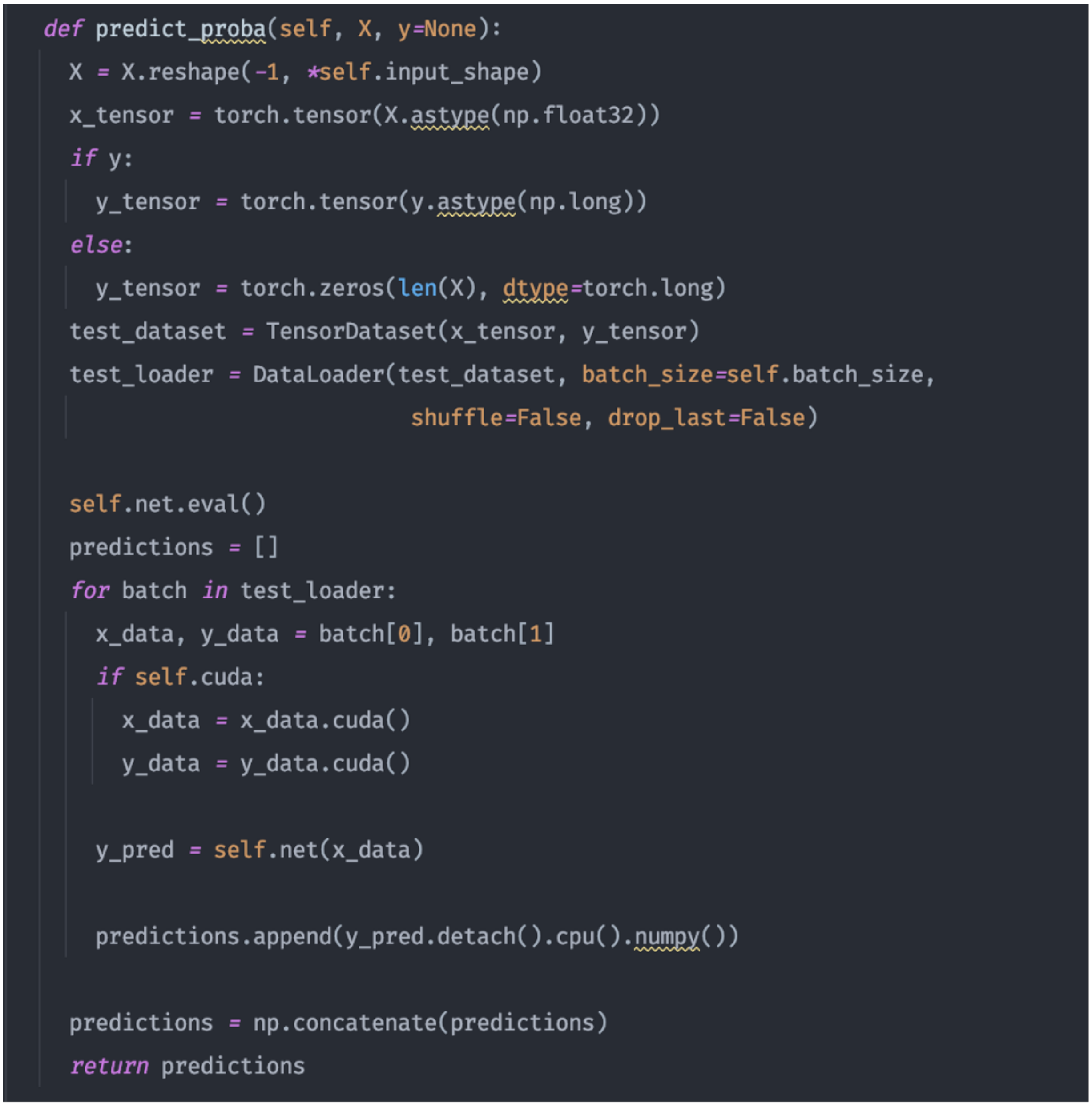
Most interesting: the fit (fig. 4.8) method. In fact, it works like training our first neural network - we skip the data, calculate Loss, and do the back propagation of the error. The only question is: how do we understand how many epochs the model needs to train? The first solution may be to assign a certain number of epochs before training. For example, 200 (as in the first model). But this can lead to overtraining, and maybe even undertraining. The second solution is to watch how the accuracy of our model changes. In the first neural network, you can see that at the end the accuracy is kept at about the same level for several epochs in a row. Then we can stop training if the model does not improve its quality for several (tolEpochs) epochs (that is, accuracy changes by no more than accuracyTol)



```
def fit(self, X, y):
    self.net = self.net_type(**self.net_params)
    if self.cuda:
        self.net = self.net.cuda()
    self.optim = self.optim_type(self.net.parameters(), **self.optim_params)

    uniq_classes = np.sort(np.unique(y))
    self.classes_ = uniq_classes

    X = X.reshape(-1, *self.input_shape)
    x_tensor = torch.tensor(X.astype(np.float32))
    y_tensor = torch.tensor(y.astype(np.long))
    train_dataset = TensorDataset(x_tensor, y_tensor)
    train_loader = DataLoader(train_dataset, batch_size=self.batch_size,
                              shuffle=True, drop_last=False)
    last_accuracies = []
    epoch = 0
    keep_training = True
    while keep_training:
        self.net.train()
        train_samples_count = 0
        true_train_samples_count = 0

        for batch in train_loader:
            x_data, y_data = batch[0], batch[1]
            if self.cuda:
                x_data = x_data.cuda()
                y_data = y_data.cuda()

            y_pred = self.net(x_data)
            loss = self.loss_fn(y_pred, y_data)
```

Fig 4.8: Fit method description

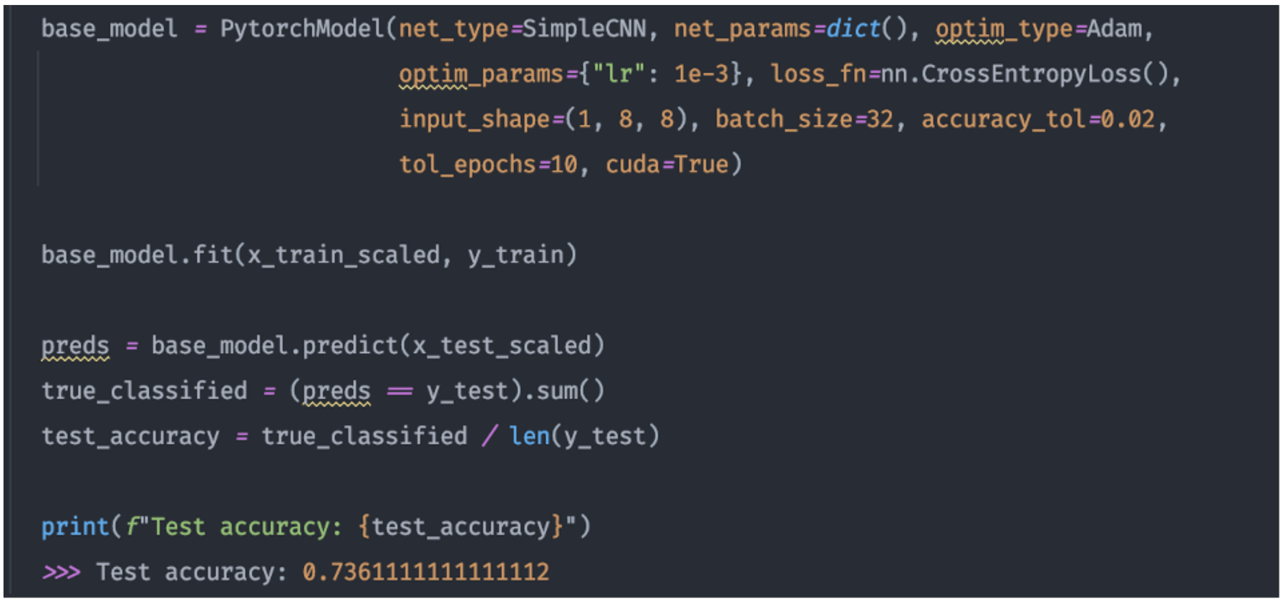
Now author will make a function to predict (fig. 4.9) the probabilities of classes. In fact, we also skip the data, just don't count Loss, but write everything into some array:



```
def predict_proba(self, X, y=None):
    X = X.reshape(-1, *self.input_shape)
    x_tensor = torch.tensor(X.astype(np.float32))
    if y:
        y_tensor = torch.tensor(y.astype(np.long))
    else:
        y_tensor = torch.zeros(len(X), dtype=torch.long)
    test_dataset = TensorDataset(x_tensor, y_tensor)
    test_loader = DataLoader(test_dataset, batch_size=self.batch_size,
                             shuffle=False, drop_last=False)

    self.net.eval()
    predictions = []
    for batch in test_loader:
        x_data, y_data = batch[0], batch[1]
        if self.cuda:
            x_data = x_data.cuda()
            y_data = y_data.cuda()

        y_pred = self.net(x_data)

        predictions.append(y_pred.detach().cpu().numpy())

    predictions = np.concatenate(predictions)
    return predictions
```

Fig 4.9: Predict method description

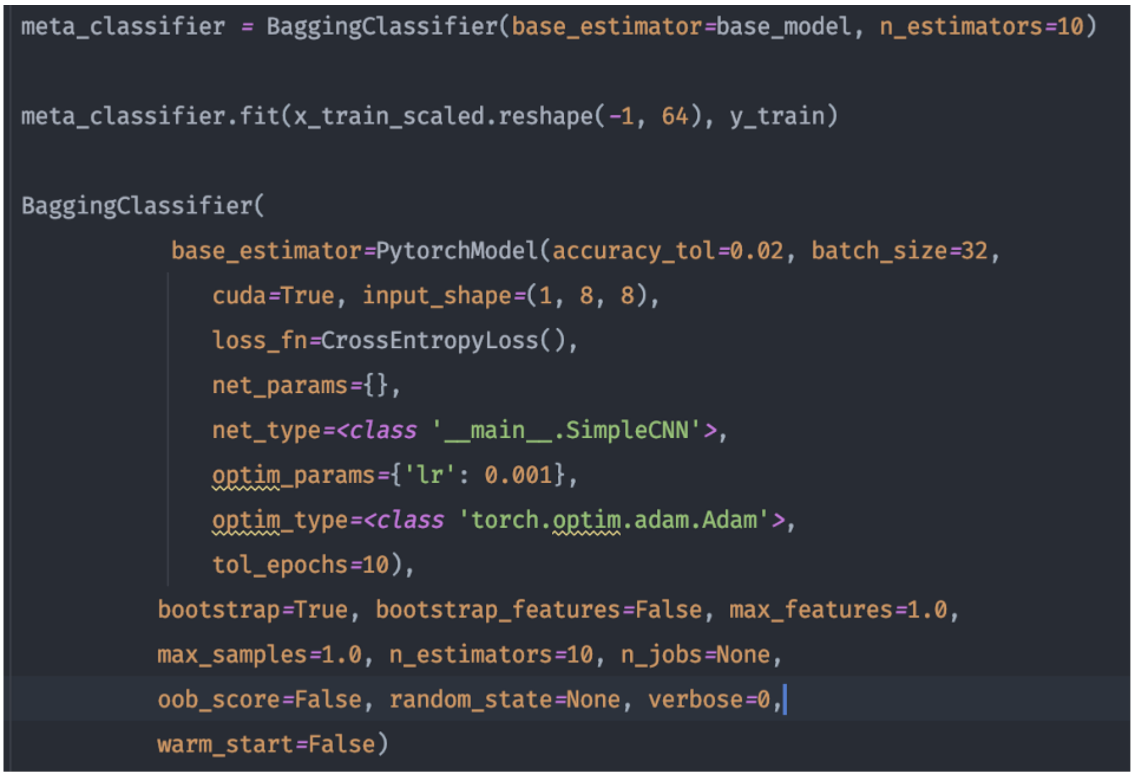
Next picture, describes how to use our neural classification network, all params were written in model constructor for simplicity, but it can be written in file or in data base configuration table, lets train our network (fig. 4.10).



```
base_model = PytorchModel(net_type=SimpleCNN, net_params=dict(), optim_type=Adam,
                          optim_params={"lr": 1e-3}, loss_fn=nn.CrossEntropyLoss(),
                          input_shape=(1, 8, 8), batch_size=32, accuracy_tol=0.02,
                          tol_epochs=10, cuda=True)

base_model.fit(x_train_scaled, y_train)

preds = base_model.predict(x_test_scaled)
true_classified = (preds == y_test).sum()
test_accuracy = true_classified / len(y_test)

print(f"Test accuracy: {test_accuracy}")
>>> Test accuracy: 0.7361111111111112
```

Fig: 4.10: Base model train code

From result in fig. 4.10 our network accuracy 73.6 percent’s. Next step, is create ensemble which consist of few simple fuzzy neural network, and learn it’s from our set of data, with help of bagging algorithm (fig. 4.11) all params were written in model constructor for simplicity, but it can be written in file or in data base configuration table, so connect all created models into a neural ensemble and learn the model with test data



```
meta_classifier = BaggingClassifier(base_estimator=base_model, n_estimators=10)

meta_classifier.fit(x_train_scaled.reshape(-1, 64), y_train)

BaggingClassifier(
          base_estimator=PytorchModel(accuracy_tol=0.02, batch_size=32,
             cuda=True, input_shape=(1, 8, 8),
             loss_fn=CrossEntropyLoss(),
             net_params={},
             net_type=<class '__main__.SimpleCNN'>,
             optim_params={'lr': 0.001},
             optim_type=<class 'torch.optim.adam.Adam'>,
             tol_epochs=10),
         bootstrap=True, bootstrap_features=False, max_features=1.0,
         max_samples=1.0, n_estimators=10, n_jobs=None,
         oob_score=False, random_state=None, verbose=0,
         warm_start=False)
```

Fig 4.11. Bagging classifier description

The next lines print result of our neural fuzzy model with bagging learning, print(meta_classifier.score(x_test_scaled.reshape(-1, 64), y_test)) >>> 0.95.

An ensemble does a better job of classification than a single model. This is due to the greater generalizing ability. Ensembles have a number of advantages over conventional designs. For example, ensembles are more robust on new data (that is, their predictions do not “jump” due to model uncertainty).

Conclusions: During the section the author considered the algorithm of training of fuzzy neural subsystem for the decision of problems of classification of images on an example of classification of pictures with numbers developed by him. The proposed algorithm can be used not only for number recognition, but also for more applied tasks, such as text and image recognition, computer vision and others. The proposed algorithm improves the result of the existing solution, which is calculated by a fuzzy neural network (classifier) TSK

CHAPTER 5
OCCUPATIONAL SAFETY

5.1. Analysis of harmful and dangerous production factors

The main production process is to develop algorithms, systems, technical documentation and write software that requires the use of computers. There are four workstations in the computer lab. They are all equipped with a PC with a liquid crystal display, and each location is connected to a local area network. A fax machine is additionally installed on the table. There are two MFPs in the laboratory. Six lamps are used for lighting. Each lamp contains two fluorescent lamps type LB-40-1. The windows of the computer lab are quite old. There is no special ventilation and sound insulation in the room. All equipment located in the computer lab is connected to a 220 V power supply.

A room with the following geometric parameters was selected for the computer laboratory: width - 4 m, length - 6.25 m, area - 25 m2, ceiling height - 3.8 m. The building and premises are constructed in accordance with the requirements [10]. The computer lab is equipped with four workstations for developers. The volume of production space for developers, operators of video terminal devices per employee is 19.5 m3, the area of the premises - 6 m2 taking into account the maximum number of employees in one shift. The plan of the computer laboratory is shown in fig. 5.1.

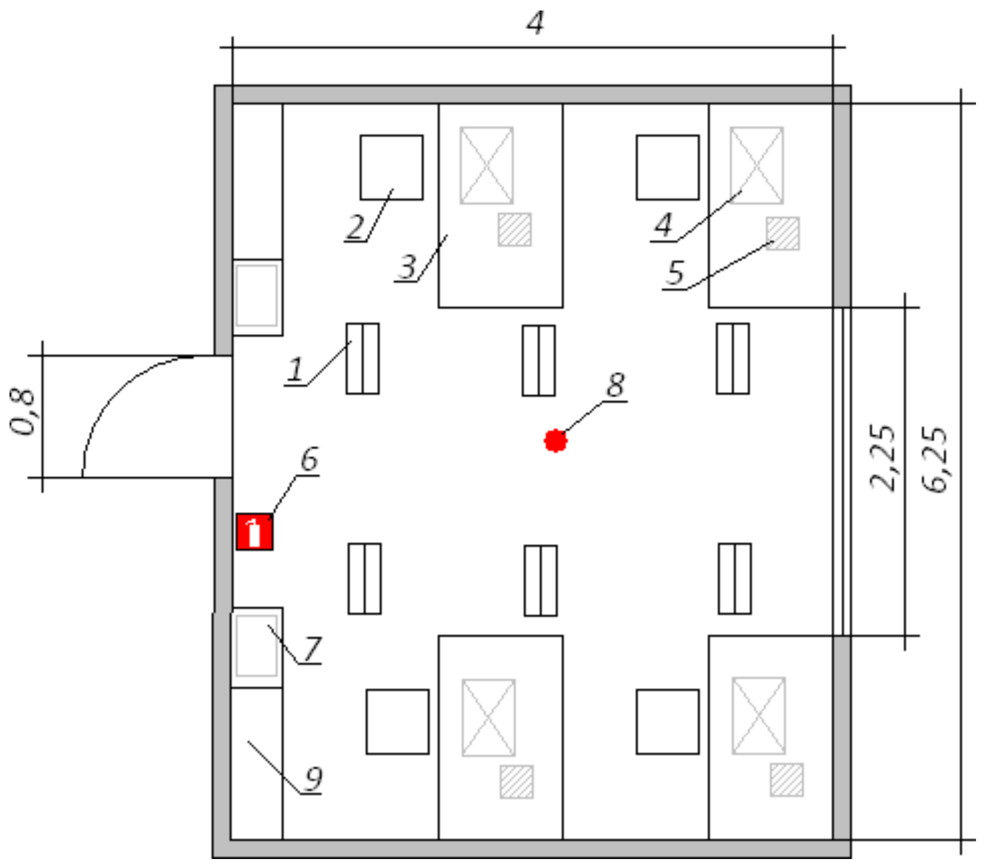


Fig 5.1: Plan of the computer laboratory: 1 - lamp; 2 - chair; 3 - table; 4 - personal computer (PC); 5 - telephone - fax; 6 - fire extinguisher; 7 - multifunction device (BFP); 8 - fire sensors; 9 - wardrobe

Workload of the neural network operator

The work of the developer is associated with a significant visual load, which requires adequate lighting. In this room, the level of natural light is sufficient, and the level of artificial - reduced. The development engineer works with computers and other office equipment, which is a source of danger of electric shock. The work of the developer is associated with a constant stay in the room, so for comfortable working conditions it is necessary to create a proper microclimate in the computer lab.

According to standard ГОСТ 12.0.003-74 the following harmful production factors can be identified that affect the employee of this computer laboratory:

1. increased noise in the workplace;
2. increased or decreased air temperature of the working area;

3. insufficient lighting of the working area;
4. lack or absence of natural light;
5. monotony of work;

Level of artificial lightning

The computer lab has six luminaires with two LB40-1 fluorescent lamps in each. The power supply of the luminaires is a 220 V mains supply.

The actual value of the illumination of this working space is only E = 210-220 lux.The category of work performed by the developer refers to high-precision work with the assignment of the category of the III century. Therefore, the normative value of the general lighting of the working space should be E = 300-500 lux. Therefore, it is necessary to take measures to increase the illumination of the room. Lighting in the developer's workplace should be such that the employee can do his job without eye strain. The calculation of the illumination of the workplace is reduced to the choice of lighting system, determining the required number of lamps, their type and location.

According to the selected category of visual works, the allowable value of the illumination of the work surface is taken E = 400 lux.

To improve the lighting of the computer lab, LED lamps will be used, namely LITWELL LED-T8S-120, the luminous flux of which is Фл = 1500лм.

Increased or decreased air temperature of the working area

Computers and office equipment are a source of significant heat, which can increase the temperature and reduce the relative humidity in the room. In rooms where computers are installed, certain parameters of the microclimate must be observed. Sanitary norms set the values of microclimate parameters that create comfortable conditions. (see Table 5.1).

Table 5.1

Optimal and acceptable parameters of the microclimate

Period of the year	Microclimate parameter	Value		
		Optimal	Let's say	In fact
Cold	Air temperature in indoors	21,0-23,4°C	23,5-25,4°C	16,1-18,0°C
	Relative humidity	40-60%	75%	35%
	Air velocity	0,1м/s	до 0,1м/s	0,1м/s
Warm	Air temperature in indoors	21,0-23,4°C	23,5-25,4°C	26,7-27,4°C
	Relative humidity	40-60%	55%	55%
	Air velocity	0,1 м/s	0,2-0,1м/s	0,1м/s

To ensure comfortable conditions, both organizational methods (rational organization of work depending on the season and the day of alternation of work and rest) and technical means (ventilation, air conditioning, heating system) are used.

The value of the actual humidity in the room in the cold period - 35%, does not fall into the range of permissible values. Therefore, in the cold season in the room you need to use humidifiers, and to increase the temperature you need to install additional heating.

In the warm season to lower the temperature you need to install air conditioning.

Increased noise in the workplace

The increased noise level in the computer lab is caused by four PCs, two multifunction devices, and the hum of the starter relay. The actual value of the noise level is 88-92 dB, when the allowable sound level is ≤ GDR, namely 50 dB. Noise measurement methods and permissible sound pressure levels in octave frequency bands, equivalent sound levels at the workplace.

Noise worsens working conditions by having a harmful effect on the human body. Workers in conditions of prolonged noise experience irritability, headaches, dizziness, memory loss, fatigue, loss of appetite, ear pain, etc. Such disturbances in the work of a number of organs and systems of the human body can cause negative changes in a person's emotional state up to stressful situations. Under the influence of noise, the concentration of attention decreases, physiological functions are disturbed, fatigue appears due to increased energy expenditure and mental stress, speech commutation deteriorates. All this reduces a person's ability to work and his productivity, quality and safety.

Additional sound insulation is required to reduce the noise level. As sound-insulating materials used in the construction of floors to reduce the transmission of structural (impact) sound are mainly used mats and plates of glass and mineral fiber, soft boards of wood chips, cardboard, rubber, insulated linoleum, as well as the replacement of windows with soundproofing.

5.2. Calculation to improve the level of artificial lighting

To improve the lighting of the computer lab, LED lamps will be used, namely LITWELL LED-T8S-120, the luminous flux of which is Фл = 1500лм.

According to the selected category of visual works, the allowable value of the illumination of the work surface is taken E = 400 lux.

To calculate the illumination of the CL, we use the method of luminous flux. To determine the number of lamps determine the luminous flux incident on the surface by formula 4.1:

$F = \frac{EkSZ}{\eta}$, where F - luminous flux, Лм

E - normalized optimal illuminance, Lk, E = 400 Lk;

S - area of the lighting room (in our case S = 25 m2);

Z - is the coefficient of minimum illumination, which characterizes the uneven lighting. Accepted at the most favorable location of the lamps, when the light flux is used to illuminate the work area most efficiently, (Z = 1.1)

k - is the stock factor, which takes into account the reduction of the luminous flux of the lamp as a result of contamination of the lamps during operation (its value is determined by the table of stock ratios for different rooms and in our case k = 1.2);

η - is the coefficient of utilization of the luminous flux from the lamp, which shows what part of the luminous flux of the lamp reaches the illuminating surface, including due to the reflection of the luminous flux from the walls, ceiling and work surface.

To determine the coefficient η you need to calculate the index of the room i by formula 5.1:

$$\boldsymbol{i = \frac{S}{h * (A + B)}} \qquad (5.1),$$

where S is the area of the room, S = 25 m2;

h - is the height of the suspension of lamps above the work surface, m;

A - width of the room, A = 4 m;

B is the length of the room, B = 6.25 m.

The height of the suspension is found by the formula 5.2:

$$\boldsymbol{h = H - h_{c\text{в}} - h_p} \qquad \textbf{(5.2)}$$

where H - geometric height KL, H = 3,8 m;

3,2 - 0,3 - 0,9 = 2 m

$$i = \frac{25}{2{,}1 * (4 + 6{,}25)} = 0{,}93$$

According to the room index and luminous flux coefficients from the floor - 10% (0.1), from the walls - 30% (0.3) and from the ceiling - 50% (0.5), we determine the value for the LED lamp LITWELL LED-T8S-120 luminous flux utilization factor η = 0.51.

Substitute all the values in formula 5.1 to determine the luminous flux:

$$F = (400*1.2*25*1.1)/0{,}51 = 25882 \text{ Лм}$$

Calculate the required number of lamps:

$$N = \frac{F}{F_L};\ N=25882/1500 = 18$$

So, for lighting we use 6 lamps, each lamp is completed with 3 lamps. Luminaires are placed in two rows, three in each row.

This room does not belong to those that need emergency lighting.

5.3. Occupational Safety Instruction

1. General labor protection requirements
   1.1. The workplace for working with video terminals must be located so that the field of view of the worker does not get windows, lighting fixtures, surfaces that have the property of reflection. The surface of the desktop should not be polished. To prevent glare on the video monitor screen, especially in summer and sunny days, the video monitor screen should be positioned so that the light from the window falls on the side, preferably on the left.
   1.2. The video screen of the PC monitor must be away from the user's eyes (hereinafter referred to as the operator)
   1.3. at a distance of at least 500 - 700 mm. The angle of view is in the range of 10-40 degrees. The most rational is the location of the screen perpendicular to the line of sight of the operator.
   1.4. The PC must be located at a distance of not less than 1 meter from the heat source.

1.5. The keyboard should be placed on a table surface or a special stand at a distance of 100-300 mm from the edge turned to the user. The angle of the keyboard panel to the horizontal surface should be between 5 and 15 degrees.

1.6. The height of the working surface of the table should be within 680-800 mm.

1.7. The chair must provide the operator with comfortable working conditions and physiological rational working posture during work. The chair must be able to adjust the height of the seat surface, the angle of the backrest and the height of the backrest.

1.8. To protect against direct sunlight, which creates glare on the video monitor screen, sun protection devices must be installed on the windows. The screen of the video monitor should be positioned so that the light from the window falls on the workplace from the side, preferably on the left.

2. Safety Requirements before starting work

2.1. Before starting work, the employee must check the integrity of the enclosures of the system unit, video monitor, printer, keyboard.

2.2. Check the integrity of power cables, their connection points (mains sockets, mains extensions, junction boxes, plugs).

2.3. Prepare your workplace by removing things that may interfere with the work.

2.4. Turn on the PC power.

2.5. If the PC does not boot after the PC is turned on or the computer does not go into operation, the employee must notify the head or specialist of the information technology department.

2.6. If damage or any other defects are found, notify the immediate supervisor. Do not start work without his instructions.

3. Safety Requirements during operation

3.1. It is necessary to stably place all the components of the device on the table, including the keyboard. However, it must be possible to move the keyboard. Its location and angle should correspond to the wishes of the PC user. If the design of the keyboard does not provide space for palm support, it should be placed at a distance of at least 100 mm from the edge of the table in the optimal area of the monitor field. When working on the keyboard, you should sit straight, do not strain.

3.2. To reduce the adverse impact on the user of devices such as "mouse" (forced posture, the need for constant quality control) should provide a free larger surface area of the table to move the "mouse" and a comfortable elbow joint.

3.3. Unauthorized conversations, annoying noises, etc. are not allowed.

3.4. Periodically, when the PC is turned off, dust should be removed from the surfaces of the equipment with a cotton swab slightly moistened with a soap solution. The screen and protective screen are wiped with an alcohol swab.

3.5. FORBIDDEN:

3.5.1. self-repair equipment in which the tube and other elements may be under high voltage (up to 25 kV.)

3.5.2. put any things on the PC hardware, sandwiches and drinks on or next to the keyboard. This can disable it;

3.5.3. cover the ventilation holes in the equipment, it can lead to overheating and failure.

3.6. To reduce the negative impact on the health of employees of various risk factors associated with work on the PC, there are additional regulated breaks for rest of PC users:

3.6.1. after each time of continuous work - 10 minutes;

3.6.2. every 2 hours - 15 minutes.

3.7. In order to reduce the negative impact of monotony, it is advisable to alternate operations of text input and data entry (change the content and pace of work), etc.

4. Safety Requirements after work

4.1. Finish and save the PC files that were in the works. Perform all steps to properly shut down the operating system.

4.2. Turn off the printer and other peripherals, turn off the system unit. If there is an uninterruptible power supply (UPS), turn off its power.

4.3. Turn off the PC with the “POWER” button and unplug the power cord.

4.4. Cover the keyboard with a lid to prevent dust from entering it.

4.5. Bring order to the workplace.

5. Safety Requirements at emergency situations

5.1. If the PC smells burnt or the metal parts of the PC are exposed to electric shock, unplug the PC immediately and notify your supervisor.

5.2. In case of fire, immediately start extinguishing with available fire extinguishing means and notify by phone 101 (city fire department) and the head of the DPD of the enterprise. Remember that extinguishing electrical installations should be carbon dioxide fire extinguishers, dry sand to avoid electric shock.

5.3. In case of injury, stop work, provide first aid, call an ambulance by phone 103, if necessary, take to a hospital.

CHAPTER 6
ENVIRONMENT PROTECTION

6.1. The impact of neural networks on the environment

The field of artificial intelligence is often compared to the oil industry: after extraction and refining, data, like oil, can become a very profitable commodity. However, it is now clear that this metaphor is expanding. Like fossil fuels, the process of deep learning has a great impact on the environment. Researchers at the University of Massachusetts warn about the shadow side of artificial intelligence in a new work: the amount of computation required to train the model requires a huge amount of energy and causes the release of carbon dioxide into the atmosphere. The figures given by the authors show that training one model costs humanity more than operating five cars.

It turned out that this process could emit more than 626,000 pounds (about 300,000 kg) in carbon dioxide equivalent, which is almost five times higher than the emissions of a typical car in five years (including the production of the car itself).

Personal computer as a source of pollution

A computer is a source of electromagnetic radiation. It is believed that electromagnetic radiation can cause disorders of the nervous system, decreased immunity, disorders of the cardiovascular system and abnormalities during pregnancy.

The biological response of a person is influenced by such parameters of computer electromagnetic fields as the intensity and frequency of radiation, the duration of irradiation and signal modulation, the frequency spectrum and the frequency of action.

Computer monitors are a source of X-rays, beta and gamma radiation. X-rays are present only when the monitor is running. The X-ray spectrum is safe with a set of monoenergetic lines. The maximum energy of the spectrum is ~ 20 KV.

Beta and gamma radiation are present when the monitor is on and off. The source of these radiations is the radioactive decay of nuclei of the uranium and thorium families, as well as potassium-40 nuclei. The spectral composition of gamma radiation mainly consists of a set of monoenergetic lines. The beta radiation of the monitor is determined mainly by the radioactive decay of potassium-40 nuclei; the spectral composition of beta radiation is continuous, and its maximum energy is ~ 1.3 meV. Under certain conditions, these ionizing radiations can harm human health, including clouding of the lens of the eye. To reduce the harmful effects of ionizing radiation, the anode voltage was reduced in the monitors, and lead was added to the monitor glass.

During operation, the computer creates an electrostatic field around itself, which deionizes the environment, and when the board and the monitor case are heated, they emit harmful substances into the air. All this makes the air dry, weakly ionized, with a specific odor and generally "difficult" to breathe. Naturally, such air can not be useful for the body and can lead to allergic diseases, respiratory diseases and other disorders.

Since the most harmful are monitors and system units, we will consider them in more detail.

There are three types of monitors:

1. Monitors based on a cathode ray tube;
2. Liquid crystal;
3. Plasma.

The main source of negative impact of the computer on the environment and user health is monitors based on a cathode ray tube.

The degree of human hazard of ionizing radiation emitted by computer monitors depends on the levels of ionizing radiation that enter the eyes of users.

X-ray exposure dose rate at a distance of 0.05 m from the screen and housing of the video terminal at any position of control devices in accordance with the Radiation Safety Standards of Ukraine (NRBU-97), approved by the State Sanitary Doctor of the Ministry of Health of

Ukraine from 18.08.97 №58, should not exceed A / kg, which corresponds to an equivalent dose of 0.1 mber / h (100 μR / h). The level of gamma radiation depends on the concentrations of natural radionuclides in the monitor glass, which for potassium-40 is 3-10%, for thorium and uranium. Based on this, we can show that at a distance of 5 cm from the monitor screen, the dose rate of gamma radiation is insignificant (~ 0.03-0.1 μR / hour) and is 0.5% of the background dose rate. Beta radiation can be easily measured with a beta counter. Such measurements show that at a distance of 5 cm from the monitor screen, the beta radiation flux density can be 0.2-0.5 parts/s.

Scintillation spectrometers with thin crystals of NA1 (T1) or Cs1 (T1) and with a sufficiently large surface are usually used to measure X-rays. The results obtained with the help of such counters show that the maximum dose rate of X-rays at a distance of 5 cm from the screen of the comparison monitor with the background and does not exceed 5-15 μR / hour.

Based on this, the power of the equivalent radiation dose for the adverse event, when the eyes of the computer operator are located at a distance of 5 cm from the monitor screen, is 0.3-0.4 μSv / hour. This result indicates the radiation safety of computer monitors, as the accumulated by the lens of the eye annual equivalent dose (~ 0.7 mSv) is 20 times less than the allowable NRBU-97 value.

Effects of electromagnetic oscillations on human health

Disorders caused by the action of electromagnetic fields on humans are manifested by higher nervous activity and bioelectrical activity of the brain. One of the reasons that contribute to the development of adverse factors in humans is the information disintegration in the brain system with subsequent violations in other functional systems. Thus, it has been established that the human endocrine, immune and reproductive systems are too sensitive to electromagnetic fields (EMF). Periodic action of EMF can lead to persistent changes in hormonal status, adversely affect genetic structures.

For the first time, a significant comprehensive study of the possible adverse effects of electromagnetic fields on the health of users was conducted in 1994 in Canada.

According to the generalized data, in computer workers from 2 to 6 hours a day functional disorders of the central nervous system occur more often on average 4-6 times than in control groups, diseases of the cardiovascular system - 2 times more often. Diseases of the upper respiratory tract - 1.9 times more often, diseases of the musculoskeletal system - 3.1 times more often. As the duration of work on the computer increases, the ratio of healthy and sick among users increases sharply.

Studies of the functional state of computer users conducted in 1996 at the Center for Electromagnetic Safety showed that even with short-term operation of 45 minutes, the user's body under the influence of electromagnetic radiation of the monitor undergoes significant changes in hormonal status and specific changes in brain biocurrents. These effects are especially bright and persistent in women. It is noticed that only about 20% of people have a negative reaction to the functional state of the body when working on a PC for more than 1 hour. In 80% it happens earlier. Therefore, methods and means should be developed to reduce the negative impact of electromagnetic fields on human health.

Basic principles of environmental protection

The main principles of environmental protection are:

a) the priority of environmental safety requirements, the obligation to comply with environmental standards, standards and limits on the use of natural resources in the implementation of economic, managerial and other activities;

b) guaranteeing an environmentally safe environment for human life and health;

c) precautionary nature of measures to protect the environment;

d) greening of material production on the basis of comprehensive solutions in matters of environmental protection, use and reproduction of renewable natural resources, widespread introduction of new technologies;

e) preservation of spatial and species diversity and integrity of natural objects and complexes;

f) scientifically substantiated coordination of ecological, economic and social interests of society on the basis of a combination of interdisciplinary knowledge of ecological, social, natural and technical sciences and forecasting of the state of the natural environment;

g) mandatory environmental impact assessment;

{Item "is" of Article 3 as amended by Laws № 3038-VI of February 17, 2011, № 2059-VIII of May 23, 2017}

h) publicity and democracy in decision-making, the implementation of which affects the state of the environment, the formation of the population's ecological worldview;

i) scientifically substantiated rationing of the impact of economic and other activities on the environment;

j) gratuitousness of general and payment of special use of natural resources for economic activity;

k) compensation for damage caused by violation of environmental legislation;

{Paragraphs "i" of Article 3 as amended by Law № 2756-VI of December 2, 2010}

l) addressing issues of environmental protection and use of natural resources, taking into account the degree of anthropogenic variability of territories, the cumulative effect of factors that adversely affect the ecological situation;

m) a combination of incentives and responsibilities for environmental protection;

n) solving problems of environmental protection on the basis of broad interstate cooperation;

o) establishment of an environmental tax, rent for special use of water, rent for special use of forest resources, rent for subsoil use in accordance with the Tax Code of Ukraine;

{Article 3 is supplemented by item "l" in accordance with Law № 2756-VI of 02.12.2010; with changes made in accordance with the Law № 71-VIII of 28.12.2014}

p) taking into account the results of strategic environmental assessment.

{Article 3 is supplemented by item "m" in accordance with Law № 2354-VIII of March 20, 2018}

{Article 3 as amended by Law № 186/98-BP of 05.03.98}

6.2. Calculation of the impact of neural networks on the environment

The paper especially considers the process of learning a model for natural language processing (NLP), a subfield of artificial intelligence, which is engaged in learning machines for working with human language. Over the past two years, the NLP community has reached several important milestones in machine translation, proposal completion, and other standard evaluation tasks.

The infamous OpenAI GPT-2 model, as an example, has succeeded in writing compelling fake news notes.

But such advances required the training of ever-larger models on stretched datasets from sentences extracted from the Internet. This computational approach is expensive and very energy intensive.

The researchers looked at four models in the area responsible for the biggest performance jumps: Transformer, ELMo, BERT and GPT-2.

They trained each of them on a single GPU during the day to measure power consumption.

They then took the number of training hours indicated in the initial documents of the model to calculate the total energy consumed throughout the training process. This amount has been translated into the equivalent of pounds of carbon dioxide, which corresponds to the energy consumption structure of AWS from Amazon, the largest provider of cloud services.

It turned out that the computational and environmental costs of training increased in proportion to the size of the model, and then increased many times as the final accuracy of the model was adjusted. Finding a neural architecture that attempts to optimize the model by gradually changing the structure of the neural network through trial and error is extremely costly with little performance gain. Without it, the most expensive model BERT left a carbon footprint of 1,400 pounds (635 kg), which is close to the Trans-American round trip.

Moreover, these figures should only be considered as baselines.

In total, scientists estimate that the process of creating and testing a final model worthy of publication required the training of 4,789 models in six months. In terms of CO2 equivalent, it is about 35,000 kg.

The significance of these numbers is enormous, especially given the current trends in artificial intelligence research.

In general, artificial intelligence research is neglected because large neural networks are found to be useful for a variety of tasks, and companies with unlimited computing resources will use them to gain a competitive advantage. But for the climate it will not be very good.

6.3. Calculation of the sanitary protection zone for the source of electromagnetic radiation

The effect of electromagnetic radiation is determined by the field strength, the duration of irradiation and the wavelength. The greater the field strength, the shorter the wavelength and the longer the irradiation time, the stronger the effect.

The creation of a sanitary protection zone (SPZ) around the emitter is one of the measures taken to eliminate the negative impact of EMF. For a powerful energy source, such a zone may consist of a strict regime zone, which is fenced and protected and the stay of people in which it is prohibited, and a restricted use zone, in which it is allowed to place warehouses, workshops, garages and other facilities hours per day. The size of the SPZ is calculated separately for each object, the main condition of such calculation is that at the boundary of the zone the parameters of the field should not exceed their MAL (maximum allowable level).

Calculations are specified by means of instrumental control measurements(table 6.1).

Table 6.1

MAL EMF for populated areas have the following values:

Frequency subband	Field frequency	Wavelength	MAL of field
Low	30-300 kGz	10-1 km	25 В/м
Medium	0,3-3 MGz	1-0,1 km	15 В/м
Hight	3-30 MGz	100-10 m	10 В/м
Very hight	30-300 MGz	10-1 m	3 В/м
Ultra high	0,3-3 GGz	100-10 sm	10 мкВт/sm^2
Extremal hight	3-300 GGz	10-0,1 sm	10 мкВт/sm^2

In the low frequency range - very high MAL EMFs are determined by the field strength, and in the ultra high and ultrahigh frequencies are determined by the energy flux density (EFD), which means the amount of energy passing through each cm2 of body surface perpendicular to the propagation of radiation.

The main measure of the EMF is the creation of SPZ around the transmitter antennas, the size of the zones is calculated so that at the boundary of the zone the field strength should not exceed its MAL.

Calculate the radius of the sanitary protection zone according to the following formula:

$$R = \sqrt{\frac{P_{cp} \cdot G}{4\pi\sigma}} = 2,27$$

where P_{cp} is the average radiation power of the source, W;

G is the gain of the antenna, equal to 10 · 103;

σ is the energy flux density, W / m2.

The average radiation power is calculated by the formula:

$$Bm_{cp} = P_i \cdot \tau_i \cdot F = 65$$

where P_i is the pulsed power of the source radiation, which is equal to 50 kW;

τ_i is the pulse duration (1.3 μs);

F is the pulse repetition frequency (1000 Hz).

The boundaries of the sanitary protection zone are determined by the allowable values of energy flux density (EFD):

a) for service personnel involved in the operation of the radiation source - 10 W / m2;

b) for personnel not connected with the operation of the radiation source - 5 W / m2;

c) for outsiders (population) - 0.15 W / m2.

So, after the calculation, we see that in this case the radius of the sanitary protection zone will be 2.27 m.

CONCLUSIONS

Among the existing methods of classifying security events, security incidents, threats, etc., intelligent information technologies are widely used - artificial neural networks (ANNs), fuzzy and neuro-fuzzy systems, evolutionary algorithms, multi-agent and immune systems. However, often due to the complexity of the problem, poor quality of training data and other reasons, it is not possible to achieve a satisfactory quality of the model. Then it makes sense to apply a set of models used together to solve a single problem. The analysis of the practical use of such systems allows us to assert that increasing the efficiency of their application is possible due to the use of several technologies within the framework of one information security (IS) system, for example, a collective (ensemble) of neural networks. An improved integrated approach to the construction of a neural network ensemble is proposed, which improves the efficiency of solving the problem of classifying security events.

As we can see, an ensemble that includes fuzzy neural networks has clear advantages. Fuzzy network ensembles return better results, which makes their prediction more accurate than in conventional neural networks or in networks based on conventional fuzzy inferences. If you use the proposed algorithm, the learning outcomes can rise to a new level. Using different approaches to model training, you can get different results. The proposed algorithm is based on bagging. An ensemble does a better job of classification than a single model. This is due to the greater generalizing ability. Ensembles have a number of advantages over conventional designs.

Enhance and increase the variance of your single score as they combine multiple scores from different models. So that as a result can get a model with higher stability.

If the problem is that one model has very low performance, then the displacement in the bags will be better. What's less, Boosting can create a combined model with fewer errors, by optimizing the benefits and reducing the pitfalls of a single model.

On the contrary, if the complexity of single models is due to them, the best option would be bagging. Reinforcement, in turn, does not help to avoid retraining; In fact, this technique itself faces this problem. For this reason, bagging is more effective than busting.

What can be improved for further research?

First of all, you can try to teach models of decision-making subsystems using other algorithms, namely: boosting and stacking. You can also try to train systems by combining these two approaches by choosing the best candidates for your ensemble. Unlike running, using a boosting model will gain knowledge of its predecessors and process them, which can improve results and give clearer predictions.Thirdly, it is possible to form an ensemble of different types of neurons. Then the difference in how the models "see" the data increases. For example, we can use different topologies of fuzzy neural networks, namely Nefclass, Anfis, etc.

References

1. Chumachenko E. I. Using ANFIS and NEFCLASS neural networks in classification problems / E. I. Chumachenko, D. Yu. Koval, G. A. Sipakov, D. D. Shevchuk // Electronics and Control Systems, N 1(43) – Kyiv: NAU, 2015. – pp. 93–98.
2. Hinton, G. E. “Reducing the dimensionality of data with neural networks” / G. E. Hinton, and R. R.Salakhutdinov. // Science. – 28 July 2006. – Vol. 313. – No. 5786. – pp. 504–507.
3. Hinton, G. E., Osindero, S. and Teh, Y. (2006). A fast learning algorithm for deep belief nets. Neural Computation, 18, pp 1527-1554.
4. Graves. 2012. “Supervised Sequence Labelling with Recurrent Neural Networks.”
5. Gers, Schmidhuber, and Cummins. 1999. “Learning to Forget: Continual Prediction with LSTM.”
6. Hochreiter and Schmidhuber. 1997. “Long short-term memory.”
7. Goodfellow et al. 2014. “Generative Adversarial Networks.”
8. Gehring et al. 2016. “A Convolutional Encoder Model for Neural Machine Translation.”
9. Nogueira dos Santos and Gatti. 2014. “Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts.”
10. Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*, (B. Schölkopf, J. Platt, and T. Hoffman, eds.), pp. 153- 160, MIT Press, 2007.
11. Wang Shitong and Korris Fu-Lai Chung, “On Least Learning Machine”, Journal of Jiangnan University (Natual Science Edition),vol.9, pp.505-510, Feb.2010
12. Ta Zhou, Fu-lai Chung, and Shitong Wang,” Deep TSK Fuzzy Classifier with Stacked Generalization and Triplely Concise Interpretability

Guarantee for Large Data," IEEE Transactions on Fuzzy Systems, vol.23,no.4,pp.813-826, August.2016

13. D. Rutkowska, "Neuro-fuzzy Architectures and Hybrid Learning", PhysicaVerlag, New York, 2002.
14. Takagi, T. and Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control, IEEE Transactions on Systems Man and Cybernatics 15 (1985), 116-132.
15. Andrew Hoblitzell, Meghna Babbar-Sebens, Snehasis Mukhopadhyay, "Fuzzy and deep learning approaches for user modeling in wetland design", 2016 IEEE International Conference on Systems, Man, and Cybernetics(SMC)
16. G.E.Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," http://learning.cs.toronto.edu, Aug.2010.
17. G.E. Hinton, S.Osindero and Y.W. The, "A faster learning algorithm for deep belief nets," Neural Comput, vol.1, no.7, pp.1527-1544, 2006
18. R. Kruse, C. Borgelt, F. Klawonn, C. Moewes, M. Steinbrecher, and P. Held, *Computational Intelligence. A Methodological Introduction*. Berlin: Springer, 2013.
19. Zhaohong Deng, Longbing Cao, Yizhang Jiang and Shitong Wang, "Minimax Probability TSK Fuzzy System Classifier: A More Transparent and Highly Interpretable Classification Model," IEEE Transactions on Fuzzy Systems, vol.23,no.4,pp.813-826, Apr.2015
20. Zhaohong Deng, Choi Kup-Sze, Fu-lai Chung, and Shitong Wang, "Scalable TSK Fuzzy Modeling for Very Large Datasets Using Minimal-Enclosing-Ball Approximation," IEEE Transactions on Fuzzy Systems,vol.19, no.2, pp.210-226, Apr.2011.
21. Shusen Zhou, Q Chen, X Wang, "Fuzzy deep belief networks for semi-supervised sentiment classification", Neurocomputing,131(2014) 312-322

22. Галушкин А.И. Теория нейронных сетей. Кн. 1: Учеб. пособие для вузов
/ Общая ред. А.И. Галушкина. – М.: ИПРЖР, 2000. – 416 с.: ил. (Нейрокомпьютеры и их применение).

23. Головко В. А. Нейронные сети: обучение, организация и применение. Кн. 4. Учеб. пособие для вузов / Общая ред. А.И. Галушкина. — М.: ИПРЖР,2001.–256с.

24. Аверкин, А.Н. Нечеткие множества в моделях управления и искусственного интеллекта / А.Н. Аверкин, И.З. Батыршин, А.Ф. Блишун и др.; под ред. Д.А.Поспелова. – М.: Наука. Гл. ред. физ.- мат. лит., 1986. – 312 с.