jamk.fi

Construction of Artificial Neural Associative System for Knowledge Representation

Krzysztof Abram

Bachelor's thesis
February 2018
School of Technology, Transport and Communication
Information and Communication Technology
Degree Programme in Information Technology

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Abram, Krzysztof Jan	Bachelor's thesis	February 2018
	Number of pages 60	Language of publication: English
		Permission for web publication: yes

Title of publication

Construction of Artificial Neural Associative System for Knowledge Representation

Degree programme

Degree Programme for Information Technology

Supervisor(s)

Lappalainen-Kajan Tarja, Huotari Jouni

Assigned by

Krzysztof Abram – own initiative.

Abstract

Artificial Intelligence (AI) is all about creating automatization of processes, which previously were considered as human abilities. Reasoning seems to be the most important one, because it can create an entirely new quality and allows to make an agent independent. However, reasoning must be based on knowledge, and then this process can obtain meaningful results. The knowledge base can be created in many ways; one popular way is to create artificial associative neural memory, which can store knowledge in synaptic connections between neurons.

AI is modern science and since its beginning, there has been a huge amount of research. There is also significant progress in knowledge representation, which can be divided into many classes, e.g. common sense, spatial, temporal representations. Each of them requires another approach and way of thinking.

The model of the associative neural memory can be applied as a knowledge system. One of well-known methods is to implement Hopfield Neural Network, thus obtaining auto-associative dynamic memory system. Memory can store a certain amount of data, which in context of neural memory shall be named patterns. The system should have the ability to learn a certain amount of patterns and later retrieve them by giving incomplete information, very often called a clue.

The system was implemented in C++ language with Hopfield Neural Network as core. As the training algorithm, the Hebbian Rule was chosen. The implementation contains many helping classes, created with the purpose to store data in computer memory and perform some operations. There are also graphics operations, which allows creating a picture from a raw binary format. This way the results can be interpreted by a human.

Keywords (subjects)

Knowledge Representation, Associative Neural Memories, Dynamic Neural Memories, Attractor, AI, RNN, Ontology, Hopfield Neural Network, Hebbian Rule, C++.

Miscellaneous

Contents

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ACLC2	A Computational Logic for Applicative Common Lisp
AI	Artificial Intelligence
ALC	Attributive concept Language with Complements
API	Application Programming Interface
BAM	Bi-directional Associative Memory
BDD	Binary Decision Diagram
BP	Back-Propagation learning rule
BSB	Brain-State-in-a-box
CNF	Conjunctive Normal Form
CNTK	Microsoft Cognitive Toolkit
CSR	Common-Sense Reasoning
DAM	Dynamic Associative Memory
DL	Description Logic
DLP	Description Logic Program
DPLL	Davis Putnam Logemann Loveland algorithm
FNN	Feed-forward Neural Network
GIS	Geographic Information System
JEPD	Jointly Exhaustive and Pairwise Disjoint
KB	Knowledge Base
LAM	Linear Associative Memory
LDF	Linear Discriminant Function
LTM	Long Term Memory
MEM	Minimal Extensional Mereology
NP	Nondeterministic Polynomial time
OLAM	Optimal Linear Associative Memory
OWL	Web Ontology Language
PDP	Parallel Distributed Processing
PTL	Propositional Temporal Logic
PTTP	Prolog Technology Theorem Prover
QSR	Qualitative Spatial Reasoning
RAII	Resource Acquisition is Initialization
RNN	Recurrent Neural Network
SAM	Simple Associative Memory
SN	Semantic Network
STM	Short Term Memory
TPS	Theorem Proving System
UML	Unified Modelling Language
VCS	Version Control System
WSP	Weak Supplementation Principle

1 INTRODUCTION

The Artificial Intelligence is considered a modern discipline of science and nowadays it plays a big role in information technology. Intelligent agents are units that can perform reasoning based on a knowledge acquired from a world in which they exist. Therefore, agents are able to think, expand their knowledge and act upon the environment. An ontological engineering is to represent abstract concepts in knowledge representation.

1.1 Artificial Intelligence Fundamentals

The first step in development of the Artificial Intelligence was taken after the II World War and the name was coined in 1956. For that reason as well, AI is a modern science within maximum of 70 years of history. The leaders like John McCarthy, Marvin Minsky, Allen Newell and Arthur Samuel will always be recognized as the founders of AI research. With the contribution made by them and their students by 1960s computers were able to learn checkers strategies, prove logical theorems and even solve the real world problems in algebra. The time until 1980s were called as an "AI Winter" due to difficulties with obtaining funding for projects. After that period research was quickly revived by the expert systems which were mostly a commercial initiative. (Crevier 1993) Nowadays terms like deep learning, machine learning and object detection are results of the AI research. (Russel & Norvig 2010, 16-28)

Neural networks are influential part of the AI development process. First attempt to create neuron based systems was proposed by Pitts and McCulloch. The idea was to use a neuron, simplified mathematical model of biological neuron as a logical gate. After a transistor had been discovered neuron no longer was taken into account and was finally rejected by engineers as the basic element of integrated circuits. The interest around neural networks has never died. Later in 1982 Hopfield created neural network system which represents memory. He was the first who fully recognized that memory is a collective process. (Peretto 1992, 6-11)

There are many definitions of the Artificial Intelligence. None of those shall be considered as a sufficient explanation of the new discipline of science. A lot of people find AI as a machine with a human's ability to think and to make decisions based on previously collected knowledge about the world. This is just part of the truth, because those might be also programs, not only machines. Following this idea AI is concerned as an entity with thought processes and reasoning. The definition must somehow combine both, a human performance and rationality.

First satisfactory definition of the intelligence was pointed out by Alan Turing in 1950s - the Turing Test. According to his definition computer will be intelligent if it comprises of the following capabilities (Russell & Norvig 2010, 2)

- *Natural Language Processing* – to enable it to communicate in spoken, human language.
- *Automated Reasoning* – to enable it to infer new information from stored knowledge.
- *Knowledge Representation* – to store the data about the surrounding world.
- *Machine Learning* – to adopt to new circumstances and guess patterns.

This is very general definition and above statements deliberately avoid a physical interactions between machine and external world. The Total Turing Test extends previous definition by adding new capabilities (Russell & Norvig 2010, 3)

- *Computer Vision* – to detect and perceive physical objects,
- *Robotics* – to update the environment e.g. move objects etc.

Those six points compose a skeleton of the true AI (ibid., 3). Below table, taken from the book (Russell & Norvig 2010, 2), represents citations which defines AI:

Table 1 - Definition of Artificial Intelligence (Russell & Norvig 2010, 2)

Thinking Humanly	Thinking Rationally
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behaviour in artifacts." (Nilsson 1998)

1.2 Agents

An agent is an entity which is able to perceive surrounding environment using sensors and reacting upon that environment through actuators. The idea of the agent is to abstract an unit with the ability to perceive and to behave. The environment might be physical world or virtual world emulated with another machine, it can be complex, like a city but on the other hand can be very simple, with a few components only. (Russell & Norvig 2010, 1-5)

The agent model giving as an example robot which has camera and several distance sensors. Robot can move forward, turn any side, rotate, go backward and so forth. The goal is to make the robot moving with a feature of avoiding obstacles. Therefore the robot moves using wheels which are actuators (wheels are updating the robot's state) and at the same time distance sensors read data from the environment – which in this particular case might be a room. The data received from sensors are processed in "robot's brain", where a logic part evaluates distances form objects situated in the room. If any of given distance is too small, robot is very close to an object, the brain gives an order, for instance, to turn left – that way avoiding obstacle.

Before defining a term rational agent, some key words must be explained. When the agent is scanning the environment, sensors are producing data by which sequence of actions are generated. Further each action causes changes in the environment's state. Thus sequence of actions is mapped into sequence of environment's states. The performance measure evaluates the desirability of reached environment's state. Rationality of the agent is purely depended on four factors: the performance measure, the actions which agent can perform, the agent's knowledge about the environment and agent's percept sequence. (ibid., 1-5)

The definition of rational agent is:

Definition: *For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*
(Russell & Norvig 2010, 37)

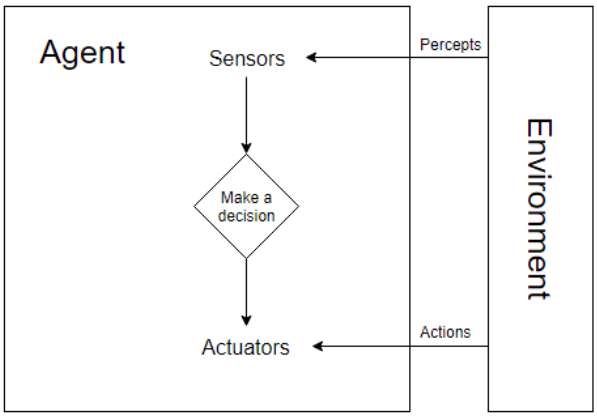


Figure 1 *- The basic concept of agent. (Russell & Norvig 2010, 37)*

1.3 Role of Knowledge and Reasoning

What is a body of knowledge? Putting this into the most simplified words, knowledge is a set of facts, premises, statements and propositions. The whole mathematics is all about that, so called logic's principles are nothing but propositions. That what we understand by the term proposition, mathematically is known as tautology, logically true sentence. An axiom is a perfect example of proposition. The axiom is logically true sentence which cannot be inferred from any other sentences. Continuing this point, the set of axioms creates the base and any other sentence can be inferred from it.

Reasoning is to posit a hypothesis or premise by asking a question and trying to prove it referring to the knowledge. Once the hypothesis is proved, it become the theorem. Therefore each theorem is the product of reasoning performed on the set of axioms.

One can ask, where all those propositions come from. The only reasonable answer for this question nowadays is to say that intelligence has created it. The intelligence embedded in agents who are able to interrogate the environment they live in. The human being. This way human has created all knowledge – algebra, geometry, physics and computer science, by perceiving the world, asking questions, proving hypothesis and making abstractions.

As the example - showing how from the set of propositions another sentences can be inferred – we can take axioms of geometry from the group I – Axioms of Connections. There are seven axioms of this group which establish a connection between the concepts like points, straight lines and planes. (Hilbert 1950, 2-3)

1. *Two distinct points A and B always completely determine a straight line a. We write AB = a or BA = a.*
2. *Any two distinct points of a straight line completely determine that line; that is, if AB = a and AC = a, where B ≠ C, then is also BC = a.*
3. *Three points A, B, C not situated in the same straight line always completely determine a plane a. We write ABC = a.*
4. *Any three points A, B, C of a plane a, which do not lie in the same straight line, completely determine that plane.*
5. *If two points A, B of a straight line a lie in a plane a, then every point of a lies in a.*
6. *If two planes a, b have a point A in common, then they have at least a second point B in common.*
7. *Upon every straight line there exist at least two points, in every plane at least three points not lying in the same straight line, and in space there exist at least four points not lying in a plane.* (ibid., 2-3)

From all above propositions two sentences might be deduced, namely theorems, and someone has done it already. (ibid.)

Theorem 1 Two straight lines of a plane have either one point or no point in common; two planes have no point in common or a straight line in common; a plane and a straight line not lying in it have no point or one point in common. (ibid., 3)

Theorem 2 *Through a straight line and a point not lying in it, or through two distinct straight lines having a common point, one and only one plane may be made to pass.* (ibid.)

1.4 Knowledge-Based Agents

Humans for whole their life acquires knowledge about the world. The environment they live in appears to be difficult in its complexity. The nature, physical existence, the body and finally the sociality all together combined make a mixture hard to understand for even thousands of units. The brain is the place where the whole knowledge is stored. Millions of sensors covering human's body, all synchronized, sends a data to the endpoint. Into the brain flows the collage of information to be interpreted, memorized and most likely used later to obtain appropriate action for the next time. The human is the true knowledge based agent.

The main part of the knowledge based agent is knowledge base. The knowledge is a set of sentences about the environment the agent exits in. e.g. that might be a set of axioms, and any other sentences can be inferred from it. Those sentences are expressed in knowledge representation language. (Russell & Norvig 2010, 235)

There is the core, where all sentences are stored, so there must be the way to retrieve information from it. There must be also the way to add new sentences to the base. There are two operation which enables querying and extending the set. Operations "ask" and "tell", respectively. Asking is the operation involving giving appropriate sentence to the knowledge base, according to which searching data will be returned. Telling on the other hand is something that can be named as learning process, which is adding new sentences. (ibid., 235)

On the figure 2, presented agent consists of a loop: environment → sensors → make a decision → actuators → environment, end so on. Each time sensors receive data from the environment, they pass it to the logic part which is responsible for making decisions. In terms of knowledge based agent this is telling the knowledge base what sensors perceive at the moment. Next thing is to ask the knowledge base what action suits to this particular situation. However this process, before the action is performed, might be much more sophisticated. Therefore the reasoning will be executed to infare necessary information about current state of the world. It must be pointed out that sensors receives just a small portion of data coming from the environment. After intensive reasoning about the outcome of many possible action sequences had been finished the final sequence must be choose. At the end the agent tells the knowledge base what action is going to be executed and agent starts turning this plan into reactions. For that purpose actuators are used. (Russell & Norvig 2010, 236)

A pseudo code presents this idea in the following way: (ibid.)

```
function agent(data) returns an Action
    knowledgeBase.tell( preparePerceptSentence(data, time) )
    action <- knowledgeBase.ask( prepareActionQuery(time) )
    knowledgeBase.tell( prepareActionSentence(action, time) )
    time.increase()
    return action
end
```

Figure 2 *- A generic knowledge-based agent procedure. Data are received from sensors.*

1.5 Knowledge Representation

Knowledge plays an important role in Artificial Intelligence, thanks to that many problems can be solved. However this knowledge must be somehow represented and stored in machine which is either a part of an agent or simulates this part.

A representation scheme is a consistent information form which explains how sentences are stored in the agent. There have been developed many kinds of schemes. Usually the form how it is designed is strongly connected with the main objectives of the project it has been developed for. E.g. some of the representation schemes are designed for learning, to acquire huge domain of knowledge in a purpose to improve decision making. (Poole & Mackworth 2010, 13-14.)

First thing to do about a problem is to give sufficient definition of it, a considered domain and a set of results. This way the problem might be treated as the relation between domain and resulting set. Sometimes is easy to guess the pattern transforming domain into codomain, but usually in real word problems, there are not satisfying answers. In order to solve a problem a solution must be defined. There are many classes of solutions. (ibid., 13-14)

An optimal solution to a problem is the best solution according to some measure of desirability, e.g. the performance measure. This kind of measure is specified as an ordinal measure. Unless if it is not too difficult to find optimal solution the utility function must be specified. Usually in real-world problems it is hard to find such solution. In order to find the optimal solution a minimum or maximum of that function must be found. In a situation when multiple criteria must be somehow combined, a cardinal measure is more accurate then. (ibid., 13-14)

An approximately optimal solution is one whose measure of desirability is close to the best solution. The advantages of cardinal measure is that it allows applying this method. In most situations agent does not need the best solution, instead of close solution can be used. This is good practice when one does not want to impact a computation performance too much. (ibid., 13-14)

There might be also probable solution, which is rather likely to be solution, but may not be actually a solution for the problem. In this approach a false-positive rate is specified - the measure how a proposition given by an agent is not correct. At the end, designers can ask the question, is solution satisficing, this is just an evaluation according to some description, how the solution is adequate. (ibid., 13-14)

1.6 Ontology

The basic concept of a discipline well known as the ontological engineering is to represent abstract concepts in knowledge representation. Most often ontology is represented by graphs. Because usually graphs are drown in a way, where the most general subjects are represented on the top and more specific appropriately under those, this concept has been called as an upper ontology. (Poole & Mackworth 2010, 437-439)

Imagine agent working in the world of humans. Amount of knowledge to be acquired by it is tremendous. Even human is not able to possess specific knowledge in each field of science and category. Usually people has general knowledge about the world, how things works, and more specific knowledge in one, two, sometimes more branches. Thus artificial knowledge-based agent will be rather modelled in the same way. (ibid., 437-439)

A general purpose ontology is applicable in a situation where no domain specific object classification will be performed. Therefore most of agents can recognize a car and they will classify it as a car, but agent with specific knowledge about motorization will be able to distinguish track from sedan car, it will be also able to tell apart brands. Nowadays general ontological engineering has so far failed, and special-purpose ontologies are used. (ibid., 437-439)

There are many well-known developed ontologies ready to be applied. The project CYC was developed to enable human like knowledge based reasoning. Free version of this project is available as OpenCyc and is designed for developers. The organization currently developing the system is named Cycorp®, their solution consists of modules like Knowledge Base, Query API, Session API. (Cycorp 2018)

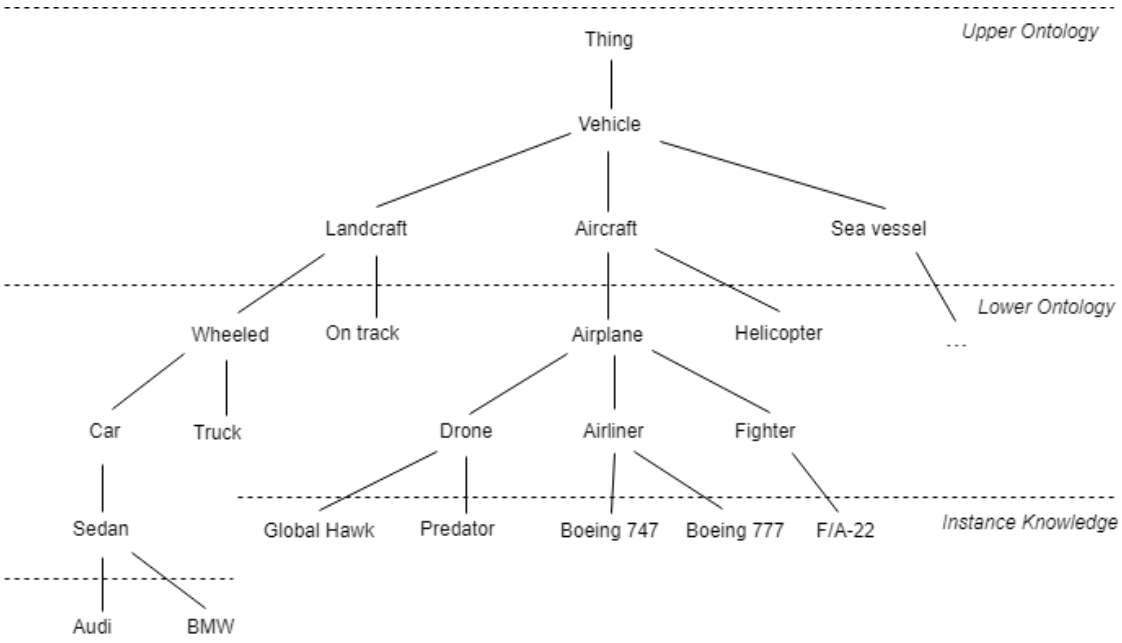


Figure 3 -The upper ontology showing how vehicles can be classified. (Poole & Mackworth 2010, 437-439)

1.7 Categories and Objects

The concept of a category refers directly to concept of a class. The class is the generalization of a set. As we know the set is a collection of objects. However there are collections which consists of too many objects, and cannot be called sets. This issue is fully described in popular Russell's paradox (Enderton 1977, 13-14) who as the first person has recognized this problem. The paradox shows that there does not exist set of everything. The most handy way to work with

categories is to treat them as sets. We can say that Tulip is a member of Flowers with a sentence tulip ∈ Flowers. If there exists some category which is a subcategory of the Flowers, e.g. roses, then we stand Roses ⊂ Flowers. (Russell & Norvig 2010, 440-442)

Categories can be organized through inheritance. One has defined a predicate P(x) = x is beautiful and then announce this $\forall x: x \in Flowers \ \& \ P(x)$ to be true. If we take now the set Roses and we know that it is the subcategory of Flowers, then from the previous sentence follows that each rose is beautiful. (ibid.)

Subsets usually are picked up from the main set according to a predicate. The subset consists of only those elements for which this predicate is true. By introducing subcategories to the knowledge representation, model can be easily ordered. The order can be called as a taxonomy or taxonomic hierarchy. Taxonomies have been used for many centuries in science, and still are used. Thanks to them we can assign a animal to appropriate category etc. (ibid.)

No one has said that two categories may not have common members. If they do not have any common members, then we can say that they are disjoint. One category can be divided into many subcategories in this way, that there is no subcategory which has any common members with other subcategories. Such a disjoint exhaustive decomposition is called as a partition. (ibid.)

Below the definition (Stewart & Tall 2015, 83):

> Definition: A partition of a set X is a set P whose members are nonempty subsets of X, subject to the conditions:
>
> (P1) Each $x \in X$ belongs to some $Y \in P$,
> (P2) If $X, Y \in P$ and $X \neq Y$, then $X \cap Y = \emptyset$

An example of above definition can be:

$X = \{1,2,3,4,5,6,7,8,9\}$
$P = \{\{1,2\},\{3,9\},\{4,6\},\{5,7\},\{8\}\}$

Furthermore the partition of the set X defines equivalence relation in the set X. (Stewart & Tall 2015, 84)

Usually in real world problems we are manipulating with a complex objects. Wheel is a part of a car, steering wheel is a part of a motorcycle and Earth is part of the Solar System. Therefore special relation can be defined. (Russell & Norvig 2010, 441)

$isPartOf(X, Y) := X \ is \ part \ of \ Y.$
e.g. $isPartOf(wheel, car)$

The isPartOf relation is transitive and reflexive: (Russell & Norvig 2010, 442)

$isPartOf(x, y) \ \& \ isPartOf(y, z) \Rightarrow isPartOf(x, z)$
$isPartOf(x, x)$

2 Methods in Knowledge Representation and Reasoning

Propositional and first-order logic play the most important roles in knowledge representation. The logic seems to be fundamental not only for whole mathematics. Conjunctive normal form appears to be good method for determining if propositional sentence is satisfiable or not. Further AI research developed techniques allowing automatically proving theorems in variety complex degree. At the same time, a lot has been done in field of representing knowledge and performing reasoning on the base. The semantic network and description logics have been employed to create appropriate syntax language to describe the knowledge and then make it possible to visualize the algorithms that do reasoning.

2.1 Propositional and First-Order Logic

For many centuries scientists have been wondering how knowledge and reasoning can be formalized. First trials in history occurred more than two thousand years ago in ancient Greece. Aristotle, Plato and Socrates were precursor using logic as the only base for reasoning. Thus they have created philosophy. Much later until XIX century there was basically no progress in the field until Frege,who is considered the father of modern logic and his influential work had critical meaning in mathematics. His ideas were later developed by Bertrand Russell, Giuseppe Peano, Charles Sanders Peirce and finally, by Alfred Tarski. (Van Harmelen, Lifschitz & Porter 2008, 4)

The logic can be divided into two the most basic sets. Fundamental one is propositional logic, which is a subset of first-order logic. (ibid.)

2.1.1 Propositional Logic

Everyone who had anything in common with mathematics used at least once in life propositional logic. As the name states the propositional logic is a branch of logic concerned with the study of propositions which are constructed from set of another propositions and connectives. At this stage, there are no predicates, quantifiers, bound and unbound variables. (Brown & Markham 2003)

Fundamental objects are *atoms*. The atom is an object that cannot be decomposed into any pieces. Simply the atom is the simplest object existing in logic. A nonempty set of atoms is usually called a *propositional signature* or *vocabulary*. Atoms are also very often called *variables*.

Right next to atoms there are *connectives*, one unary ¬ and binary like ∧,∨,⇒ and ⇔. In logic a set consisting of two elements which is τ = {true, false} is a value set named *truth values*. A function I transforming the propositional signature σ into the set τ is called an *interpretation*. A formula F is formed with atoms belonging to σ and connectives. Therefore, the value of the formula depends on interpretation. If the set of atoms is limited then count of interpretation is also limited. (Van Harmelen, Lifschitz & Porter 2008, 4)

The formula p ⇒ q has two atoms {p, q} and one connective. There are 4 possible interpretations, limited number of interpretation can be presented in a form of *truth table*: (Van Harmelen, Lifschitz & Porter 2008, 5)

Table 2 - The truth table for ⇒ connective. (Brown & Markham 2003)

p	q	p ⇒ q
false	false	true
false	true	true
true	false	false
true	true	true

For each formula, the interpretation can be specified. However there are some rules which can be applied operating with function I, all of them should be obvious for reader: (ibid.)

- $I(\bot) = false$ and $I(\mathrm{T}) = true$,
- $I(\neg F) = \neg I(F)$,
- $I(F \otimes G) = I(F) \otimes I(G)$

Where ⊗ is any of binary connective. It can be said that if $I(F)$ is true than the interpretation I *satisfies* F. (ibid., 5)

In case when every possible interpretation satisfies formula F, the formula is named *tautology*. Example of use tautology can be found in equivalence definition. Thus, two formulas are equal to each other if they are satisfied by the same interpretations. In other words if F is equivalent to G then $F \Leftrightarrow G$ is tautology. (ibid., 5)

There can exists interpretation for a set X of formulas, which satisfy each of them, then it can be said that X is *satisfiable*. (ibid., 5)

Propositional logic can be used to represent knowledge. In order to achieve it the propositional signature must be chosen in such a manner that all possible interpretations of σ correspond to states of system which is designed. A set of formulas can represent the whole knowledge base. (ibid., 5)

To clarify what propositional logic is, some examples are included below: (Brown & Markham 2003)

$$p \vee \neg p$$

$$\neg (p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$$

$$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$$

$$(p \Rightarrow q) \Leftrightarrow \neg (p \wedge \neg q) \Leftrightarrow (\neg p \vee q)$$

Above propositions can be used in programming as a knowledge base for expanding or collapsing logical expressions. An interpreter of newly defined "logic based" language can simply take a part of syntax e.g. $(p \vee q) \wedge (p \vee r)$ and replace it with $p \vee (q \wedge r)$ to make it shorter. Reverse transformation can be done as well. (Van Harmelen, Lifschitz & Porter 2008, 6)

2.1.2 First-Order Logic

First-order logic has been developed to enable use of formalization to form axioms. The foundations of it were developed by Frege and Peirce. This time, the operation is not only on set of atoms which can be arranged in any combination using connectives. First-order logic introduces new abstract objects like *predicates*, therefore this logic is also called *predicate logic*. In propositional logic set of variables – propositional signatures can have values from the set τ. In first-order logic as a value of variable any non-logical object can be used. It is said that predicate logic uses quantified variables over non-logical objects. (Hilbert D & Ackermann 1950)

A signature this time is has been extended and consists of two kind of symbols. *Predicate constants* and *function constants*. To each symbol nonnegative integer named *arity* can be assigned. This number characterize the symbol telling dimension of a domain of functions i.e. a number of arguments which function takes. Therefore function constants of arity 0 are functions with void list of arguments and are called *object constants*. The same thing can be applied to predicates constants and it is called *propositional constants* if arity is equal zero. (Van Harmelen, Lifschitz & Porter 2008, 8.)

Elements of a sequence of symbols e.g.: $a, b, c, \ldots, a_1, b_1, c_1, \ldots$ are named *object variables*. Both signature σ and function constants of σ form terms. A predicate $P(x_1, \ldots, x_n)$ of arity n and each of x_i - term of σ, form and expression which is widely used as *atomic formula*. Properly operating with connectives, atomic formulas and terms, *formulas* can be created and use of *quantifiers* be made (ibid., 8)

There are two quantifiers, one is named *general* and is symbolically presented as $\forall$. The second kind of quantifier is *existential* $\exists$. Applying quantifiers is intuitive and the underlying idea of using it is clear and ordinary. If one wants to express a thought that for each element in set A some predicate $P(x)$ is true, then can use sentence like this: $(\forall x)\ (x \in A \wedge P(x))$. Similar sentence can be formed with existential quantifier but this time the meaning will change. Therefore if we take general quantifier and replace it with existential one, like this: $(\exists x)\ (x \in A \wedge P(x))$, then it stands that there exists such object belonging to set A for which predicate P is true. (ibid., 8)

It is provided that if for each object in set A predicate P is true then the expression with existential predicate is true. Previous thought can be shown in form of the tautology: (Hilbert D & Ackermann 1950)

$$(\forall x)\ (P(x)) \Rightarrow (\exists x)\ (P(x))$$

Above expressions variable x is *bound*, this means that simply belongs to the formula. For a predicate $R(x, y)$ and expression $(\forall x \in A)\ (P(x, y))$ variable y is *unbound* or *free*, because it does not occur with any other quantifier. A formula without any free variable is *closed*. The

sentence with formula Q, $(\forall x_1, \ldots, x_n)\ Q$ is *universal closure* if $x_1, \ldots, x_n$ are free variables. (Van Harmelen, Lifschitz & Porter 2008, 9)

Below some other examples of famous and well known sentences in first-order logic: (Hilbert D & Ackermann 1950)

$$\neg\,(\forall x)\ (R(x)) \Leftrightarrow (\exists x)\ (\neg\, R(x))$$

$$(\forall x)\ (R(x) \wedge P(x)) \Leftrightarrow [(\forall x)\ (R(x)) \wedge (\forall x)\ (P(x))]$$

$$(\forall \varepsilon > 0)(\exists \delta > 0)(\forall x \in R)(\forall h \in R)\ (|h| < \delta \Rightarrow |f(x) - f(x + h)| < \varepsilon)$$

$$(\forall x)(\forall y)\ (R(x, y)) \Leftrightarrow (\forall y)(\forall x)\ (R(x, y))$$

Can quantified elements of the set be relations or other sets? No, if we use first-order logic. The problem in mathematics is solved by introducing *higher-order logic*. Basically to quantify over sets or relations we need to make use of *second-order logic*. (Hilbert D & Ackermann 1950)

Below example of expression taken from the Zermelo-Fraenkel's axiom system, this one is named *Axiom of Extensionality* and quantify over sets: (Enderton 1977, 17-18)

$$(\forall A)(\forall B)\ [(\forall z)\ (z \in A \Leftrightarrow z \in B) \Rightarrow A = B]$$

The axioms tells that for any two sets if each element belonging to set A belongs also to set B, they have the same elements, implies that these sets are equal. Another conclusion is that if there are sets that have the same elements, then set A and set B are the same set with only different name. Is like having one set and two references pointing on this set. (ibid.)

Propositional and first-order logic seems to be the very foundations of the mathematics and the amount of knowledge to be presented can be devoted to writing a book.

2.2 Automated Theorem Proving

The goal of automated theorem proving is to develop the techniques that enable fully automated or human guided searching of the proof for the hypothesis. A definition states: (Van Harmelen, Lifschitz & Porter 2008, 18)

Definition: *Automated theorem proving is the study of techniques for programming computers to search for proofs of formal assertions, either fully automatically or with varying degrees of human guidance. This area has potential applications to hardware and software verification, expert systems, planning, mathematics research, and education.* (ibid.)

The idea is having a set A of axioms and some logical consequence B, the program should be able to construct a proof starting with a set of propositions and ending with the consequence. There are many approaches to achieve this. For example, the program can search in a domain of all possible proofs as brute force search, until it finds the result. However, this method is inefficient. (ibid.)

First, automatic theorem proving was based on Herbrand's theorem (Van Harmelen, Lifschitz & Porter 2008, 19), which basically consists of enumerating test sentences against the truth using first-order logic principles. Later, it has turned out that Skolems's (ibid.) function and *conjunctive normal form* expressions might be used. All such and similar methods were a way too inefficient. In the early 1960s Robinson developed a method which later resulted in being an advance in first-order theorems proving called *resolution* method. There were also solutions using *unification algorithm* or *inverse method* competitive with the resolution. It turns out that many scientists have started to think that very likely there might not exist a unified method for proving any theorems. Because of the fact, specialized theorem provers capable to prove theorems in a limited domain were developed. For example, they resulted in creation of an *expert system*. In parallel, the previously mentioned methods were further developed due to better implementation of algorithms and data structures. It was also possible to improve those techniques because of a progress in hardware layer. Prolog Technology Theorem Prover (PTTP) was the first officially known solution refering to those improvements. Further development resulted in creation of the Prolog programming language, mostly for logics use. Some other processes have resulted in the development of popular techniques in computer science, for instance binary decision diagram known as BDD. (Van Harmelen, Lifschitz & Porter 2008, 19)

At the same time provers dealing with higher-order logic were developed. Example is TPS prover, NqTHM and ACL2 provers where Boyer, Moore and Kaufmann were persons involved in creating these. For some of them it was significant to use mathematical induction. The problem of all those solutions was the computation power in that time. Later after general purpose programming language like C have been defined and implemented, theorem proves had increased the performance and quality a lot. (Van Harmelen, Lifschitz & Porter 2008, 20)

All this effort devoted to creating universal theorem provers nowadays is a part of something that is called by its professional name *standard AI tool kit*. E.g. CNTK, TensorFlow, Keras, etc. (ibid., 20)

In order to prove if sentence S is valid or not, some method must be applied. Usually in mathematics instead of validity, satisfiability is tested. The problem of checking whether an expression is satisfiable is a class of NP-complete problems. The easiest way to achieve success is to check every possible combination of values for variables in the sentence. However, this approach has 2^n possible combinations if n is count of variables in the sentence. This way for two Boolean variables there are four combinations, for 8 variables 256 combinations and for 100 variables 2^{100} which is around 1.27×10^{30} combinations. Maybe it is better to imagine that if it is assumed that one combination can be checked in one nanosecond (10^{-9}s) then 1.27×10^{21} s is needed, which is around 40 trillion years. Does anyone want to undermine the inefficiency of this method? It must be pointed out that there were only one hundred variables. (ibid.)

Nevertheless, there is a well-known idea how to check the satisfiability for any propositional sentence. The sentence must be presented in *conjunctive normal form (CNF)*. This form consists of atom (variable) or its negation, which is literal: a clause is a disjunction of literals. Conjunctive form is when the formula consists of multiple clauses connected with conjunction. (Van Harmelen, Lifschitz & Porter 2008, 23)

Below is an example:

$$(p \vee \neg q \vee r) \wedge (q \vee \neg r) \wedge (\neg p \vee q)$$

If one thinks long enough then one will figure out the easiest way to tell quickly if a sentence is tautology or not. If in clause occurs literal and its negation, e.g.: $p \vee \neg p$ or $q \vee \neg q$ which are always true regardless of value of atom p and q, then the entire clause is true, e.g. $p \vee q \vee \neg p \vee \neg r$ is always true because it contains $p \vee \neg p$. Now it is easy to tell if such a disjunction occurs in each clause, then the whole sentence is true independently of the interpretation. (Hilbert & Ackermann 1950, 5-44)

The best way how to understand what is a conjunction's normal form is to take a propositional principle and try to express it in that form. An example of one famous example follows: (ibid.)

$$[(p \Rightarrow q) \wedge (q \Rightarrow r)] \Rightarrow (p \Rightarrow r)$$

$$(p \Rightarrow q) \Leftrightarrow (\neg p \vee q)$$

From the first and second sentence it can be concluded by replacing particular parts that:

$$[(\neg p \vee q) \wedge (\neg q \vee p)] \Rightarrow (\neg p \vee r)$$

Applying the same rule there is:

$$\neg [(\neg p \vee q) \wedge (\neg q \vee p)] \vee (\neg p \vee r)$$

By applying multiple time the De Morgan's rule (ibid.)

$$\neg (\neg p \vee q) \vee \neg (\neg q \vee p) \vee (\neg p \vee r)$$

$$(p \wedge \neg q) \vee (q \wedge \neg p) \vee (\neg p \vee r)$$

After applying distribution of conjunction over disjunction, the sentence results with:

$$(p \vee q \vee \neg p \vee r) \wedge (p \vee \neg r \vee \neg p \vee r) \wedge (\neg q \vee \neg r \vee \neg p \vee r)$$

As can be seen in each disjunction of the sentence, there are literals and their negations. This is shortly how resolution method works. (Van Harmelen, Lifschitz & Porter 2008, 22-23)

Proving the first-order theorem seems to be more complicated. These kinds of provers are based on resolution or have several analogues to this method. However, the resolution here is less efficient then DPLL procedure. (Van Harmelen, Lifschitz & Porter 2008, 25)

The Cyc project has also modules, which allows automated theorem proving. The organisation has developed their own language called CycL allowing to present and construct higher-order logic concepts. (Cycorp 2018)

2.3 Semantic and Logic in Reasoning

Categories can be treated as classes, which define variety of types. The class is generalization of a set. A comfortable way to operate on categories is to use a set theory. Categories are the most basic blocks when building the knowledge base. A tree of categories is the upper ontology,

which shows the taxonomic order between categories. Mentioned order is a kind a relation that tells that a subcategory is a subset of category. (Russell & Norvig 2010, 453-454)

The problem is not only how to organize categories, there are also difficulties with reasoning using it. Systems have been developed to provide visualization for a knowledge base and algorithms, which are able to make some inference on categories known as *semantic network*. There is also a language allowing to construct description for categories enabling methods and algorithms to define relation subset-superset between categories named *description logics*. (ibid., 453-454)

2.3.1 Semantic Network

The idea of *Semantic Network (SN)* is the ability to represent objects, categories and relations between objects, categories and objects, category and another category on a graph. This time it is more than showing relation "is subset of" between categories, the graph shows also instances, instance fields, relation between them. The most common relations are *isMemberOf* linking instance of category with the category it belongs to. Relation *isSubsetOf* is relation between two categories like subset-superset relation. There might be also relations between instances. (Russell & Norvig 2010, 454-456; Van Atteveldt 2008, 51-54)

In Figure 4 a semantic network for vehicles can be seen. This graph focuses mostly on a car branch and represents some relations between nodes of this graph. It is quite easy to mismatch relation between categories and relation between instance and category. As an example of this issue we can point out the instance "Audi A6" which is in relation *isBrandOf* "Audi". We cannot link "Four-Door" category with the "Audi" brand because category has not any brand. Therefore we can assign instance to the brand, but not the whole category. (Russell & Norvig 2010, 454-456)

In semantic network it is usually convenient to use *inheritance* for categories and perform reasoning. An example will clarify the problem. Category "Sedan" inherits number of wheels from "Car", and "Four-Door" inherits this number again from "Sedan", this way we know that "Audi A6" has 4 wheels. It usually happen that category can inherit form multiple other categories. (ibid.)

What if someone remake its car and removes one door from the model "Audi A6"? Then number of doors for this particular car will change. The problem is that this car is linked with "Four-Door" category, so how it can have three doors. In such situations the only way how to preserve the structure is to use *exception*. This one instance *overrides* number of doors, which occurs as the exception in this category. (ibid.)

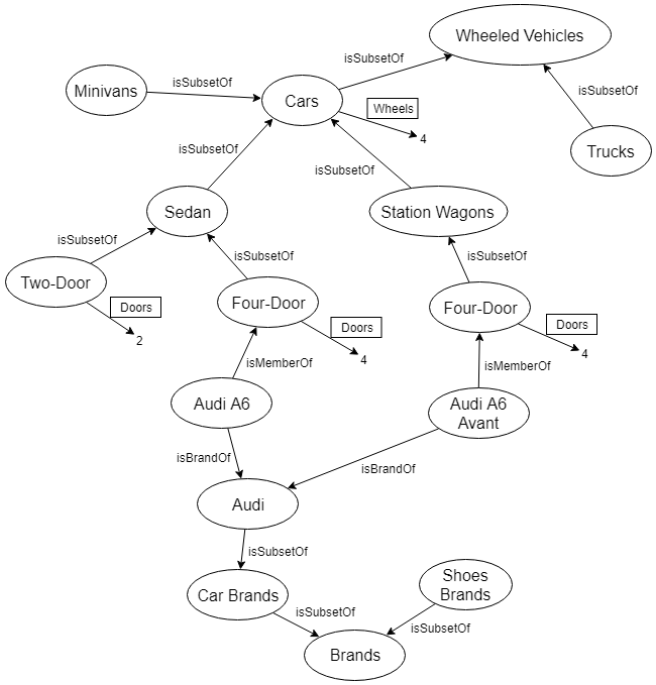


Figure 4 - The semantic network showing knowledge about vehicles and relations between them. (ibid.)

2.3.2 Description Logics

Description logics (DLs) have evolved from the semantic network and their purpose is to simplify describing objects and categories. The ultimate goal was to find a language to represent the knowledge of a given domain. One thing which description logic was developed for was to check if one category is a subcategory of another. There are two ways to achieve it. One way is to check if each member of one category belongs to another; nevertheless, this method is inefficient as it is to be useless. The second method is to compare definitions of those categories. This procedure is called *subsumption*. There is also *classification* consisting of checking if a given instance belongs to a particular category. To check if membership criteria are logically satisfiable, a *consistency* might be involved. (Russell & Norvig 2010, 456)

The first mention of description logic appeared between 1965 and 1980. The phase 0 was also called pre-DL. Because of the lack in semantics, the next steps have been taken. Later, between 1980 and 1990 it was time for phase 1. All system developed in this stage like KL-ONE, KRYPTON and LOOM made use of *structural subsumption algorithms*. The problem which occurred was that for more complex and expressive syntaxes algorithms were not able to properly perform subsumption procedure. The most popular system taking care about the limitations and issues in the phase 1 was CLASSIC. (Van Harmelen, Lifschitz & Porter 2008, 135-140)

Phase 2 between 1990 and 1995 started with a new algorithm paradigm introduced into DLs, tableau based algorithms. This family of algorithms can evaluate the consistency of the model by decomposing the concepts in the knowledge base. (ibid.)

Between years 1995 and 2000, there was a new phase 3. In this period implementers emphasize difficulties of inference procedures for expressive DLs. Such systems like FaCT, RACE and DLP were highly optimized showing that tableau based algorithm led to good behaviour of

the system. It also turned out later that good results were achieved even by applying such methods on large knowledge bases.

Today is phase 4. All methods, algorithms and approaches were used to build complex and finite industrial DLs. A popular example of it might be *Semantic Web* where *Web Ontology Language* (OWL) was developed for this purpose to represent knowledge. (Web Ontology Language 2018)

Representing knowledge in medicine is especially difficult. However, there are working projects like SNOMED-Reference Terminology, Medical Entities Dictionary and OpenGALEN. (Baader, McGuinness, Nardi & Patel-Schneider 2002, 416-417)

The *ALC* is one of the basic DL and its name stands for "*Attributive concept Language with Complements*". The syntax and semantics can be composed of constructors i.e. conjunction negation etc. (Van Harmelen, Lifschitz & Porter 2008, 140)

2.4 Reasoning with Default Knowledge

What if one instance of given category has slightly different property, e.g. 3 wheels instead of 4. Does this difference make instance to belong to another class? Not necessarily. Some instances might violate the *monotonicity*. Nonmonotonic logic has been design to deal with a situation when new information arrive over time. (Russell & Norvig 2010, 458-460)

One of the mentioned logic *circumscription* tries to make a very narrow assumption by adding extra expression as a part of sentence that are true only for particular instances. An example follows: (ibid.)

$$Plant(x) \wedge \neg Artificial(x) \Rightarrow Grows(x)$$

The above idea expresses that if x is a plant and is not artificial, then it can grow. The underlying problem appears when someone states that plants made of plastic, which usually serves as decoration, are named plants in general. Can an artificial thing, made of plastic material, grow? If we delete this part of sentence occurring in conjunction $\neg Artificial(x)$ then someone can conclude that yes, the pseudo-plant can actually grow. (ibid.)

The second logic named *default logic* is another way to deal with nonmonotonic conclusions. *Default rules* are written to make nonmonotonic conclusions: (ibid.)

$$Plant(x) : Grows(x) \,/\, Grows(x)$$

This sentence stands that if $Plant(x)$ is true and if $Grows(x)$ is consistent with the knowledge base than $Grows(x)$ can be concluded by default. In general, the rule can be shown as

$$Q : P_1, \ldots, P_n \,/\, C$$

Where Q is the prerequisite, C is the conclusion and P_i are justifications. If any of justification is false than the conclusion cannot be made.

3 Classes of Knowledge Representation

Knowledge is a set of information stored as sentences which describing the environment. However, there is no unified form to represent knowledge about every possible domain. Therefore, there are plenty of knowledge classes and performing reasoning in each of them seems to be different. Thus, physical reasoning can be somehow optimized and simplified by employing qualitative approach instead of quantitative. There might be situations when time plays significant role, in such cases even logic can be redefined and adjusted to the knowledge domain, by introducing special operators and set of rules called axioms. Extremely important is reasoning about a space, this can improve how agents perceive the world and how it reacts upon the environment.

3.1 Common Sense Reasoning

An intelligent agent has multiple sensors by which can examine the world. Each time it receives another portion of data it has to be able to understand it, change state of itself and give appropriate commands for actuators. Nevertheless, such creature exiting in physical world, e.g. it can be a robot, should know the relations between its reaction and a physical results of the reaction. Performing this kind of reasoning the agent can understand physical world much better and this way can improve the abstract model. The model includes all learned physical rules which can be applied to variety of situations. (Van Harmelen, Lifschitz & Porter 2008, 597)

Almost every physical model discovered by human has been created with the view to get as much accurate results as it is possible. And now at the same time there is requirement to make models which gives only qualitative description of physical phenomena. Usually the purpose of running any calculations involving physical rules is to give clear answer about the state of a system where predictions are made. Physical reasoning accepts physical quantities as they are discrete, so calculations can be made by dividing time and space into physically significant intervals and fragments. On the other hand there are situations were *ad hoc* physical theories are used. However to distinguish them from regular theories it must be pointed out that they are used rather for planning and reasoning. A generality is important word here. Agent will deal with variety of different environments, it must be prepared for many possible configurations of a domain. Reasoning treat time, space and objects continuously. (Van Harmelen, Lifschitz & Porter 2008, 597-598)

To physical reasoning two different nonmonotonic methods can be applied. The first is *closed-world* assumption, that whole possible entities existing in the system are known and can be easily determined. This kind of approach can be applied both in the theory level and in the specific situation. The second is an *idealization* assumption, where the model of physical world

is ideal, e.g. the ball is ideally spherical, what in the real world is rare as to be impossible. (Van Harmelen, Lifschitz & Porter 2008, 598-599)

3.1.1 Qualitative Calculus

The qualitative calculus is all about representing and reasoning how human does. Human do not discretize the space and time, instead of symbolic method is used to represent things as they were continuous. Thus, qualitative calculus is modelling of physical system in a human like manner. There exists principles that makes it even more clear. (Van Harmelen, Lifschitz & Porter 2008, 361-364)

Discretization is one of the principle consisting in quantizing continuous properties. It turns infinity of possible values, infinite domains, into finite set of values, which might be enough to represent the problem. Discretization provides first, turning a continuous properties into entities. Second, abstraction layer, which is crucial, because usually modelling has to work also in situations where only few details are known about the environment. Relevance is another principle which deals with constructing qualitative values to be relevant for a given task or situation. To tell if water is drinkable we have to measure a temperature, and if the temperature is within the range 0 to 70 Celsius degrees then one can drink it. (ibid., 361-364)

The *scenario* is the input description of the situation. A *model fragments* are various identified objects or phenomena occurring in the environment being represented as knowledge. All of them are stored in *domain theory*. The process called *model formulation* is reasoning how the scenario is constructed from model fragments. (ibid.)

3.1.2 Architecture

Definition: *An architecture for physical reasoning is a representational schema; that is, it is a structure that defines a high-level ontology and a basic set of relations and that support the representation of various general domains and of specific problems, and the carrying out of particular types of inferences over those representations.* (Van Harmelen, Lifschitz & Porter 2008, 600)

Component model and *process model* are architectures the best established and widely studied for physical reasoning. (ibid.)

Component model is the architecture where crucial part of the design are reusable components. The component is the simplest entity having a certain number of ports. To each port a count of parameters are associated. The component is an entity, which imposes some constraints on the values. Because system expands with a time, those values are usually function of the time. Mentioned constraint are either algebraic or differential. E.g. resistor has two ports, endings which are used to attach component to a circuit. To each of the port two values are assigned. One of it is current and the second is voltage. There exist some relations between input and output current. The same is with the voltage. These relations are described by equations. There might be more than one port collected together, then in this situation there is a node. Each node should have a description – what is happening there, what are particular values of assigned qualities. A system is a collection of components. (ibid., 600)

3.1.3 Domain Theories

In physical reasoning is important to understand kinematics and dynamics of objects so-called rigid. These objects are solid, closed. The shape of rigid object is constant over time – constant shape feature. Kinematic theory of those objects obviously contains much less information than dynamic, it does not elaborate on forces applied to them, and how those forces can change their state. The position of object is continuous function of time. If we have more than one object than for each two of them cannot overlap. The space they occupy is exclusive. Configuration is the state of the system where each object has its position. It is said that given configuration is *feasible* if objects do not overlap. The configuration is attainable if configuration can be change by moving some objects in the system without causing it to overlap. The fact why two objects cannot overlap has a reference in the physical world. (Van Harmelen, Lifschitz & Porter 2008, 602-608)

To create real *common-sense reasoning (CSR)* the dynamic theory must be employed. This time theory take into account also forces which cause the motion. Kinematics cares only about describing displacements, rotations and any other motions as a function of time. Many centuries ago, Newton showed how force and mass are related in motion. That discovery changed the world forever. The second Newton's law states that acceleration a of the object with a mass m is the result of applying the force $F = am$. However, this equation is true only if objects are treated as infinitely small – points of a mass. For the rigid objects, physics has another analogous theory. (ibid.)

There already exists some well-known AI programs, which can do physical reasoning for us. One of the program is named after Newton, the NEWTON program. This program can deal with qualitative computations for predicting the behaviour of objects in the system. (ibid.)

3.2 Temporal Knowledge and Reasoning

Temporal reasoning is the reasoning involving time. Because of many differences in knowledge representation and reasoning another structures for a modelling has been created. The approach has changed and the description language has changed as well. This time language has to focus on temporal representations and all of the properties related to that. Finally, both language and structures can be applied for the temporal reasoning. (Van Harmelen, Lifschitz & Porter 2008, 513-514)

3.2.1 Structures

The first thing to consider is how time should be represented in a model. One known approach is to make discretization and refer to moments in time as they were only points. Then these points must somehow be related to each other – the set of "ticks" has an order according to which two points are distinguishable. *Propositional Temporal Logic (PTL)* has been developed in order to properly describe properties based on that model of time in the system. Another option is to think about time as sequence of intervals. The interval is specified period of time with a given duration. This approach requires a bit different thinking about action in the system.

Represented properties stay true during that interval, otherwise, either the intervals are too long or the description is wrong. (Van Harmelen, Lifschitz & Porter 2008, 514-515)

Discrete based temporal logic can be defined as a tuple $\langle S, R, \varepsilon \rangle$. S is the set of all time points, R is relations between points, most often an order defined in the S set. And ε is a function mapping each point to the truth value. For point-base time representation instead of constructing a new set S, we can take *Natural Numbers*. What will happen if instead of taking discrete set we take the Real Number set. This set is known to be dense in a sense that if one take any two numbers, then for sure between them exists at least another number. This kind a set for time representation requires specific temporal language. (ibid., 515-516)

Real numbers add extra complexity to the model. Between any two time points there are infinite number of other points. The model can be described as arbitrary *granularities*. (ibid., 516)

3.2.2 Language

To appropriately describe temporal properties there is need to apply language that can easily handle the complexity of temporal reasoning. The modal logic has its origins in modal and tense logics. (ibid., 520)

The elementary pieces of any logic are connectives and operators. Temporal logic make use of tense operators trying to abstract statements about time: (ibid.)

> □ p – *p is always true in the future*,
>
> ◊ q – *q is true sometimes in the future*,
>
> ○ r – *r is true at the next moment in time*,

Two of them, ◊ and □ are analogous to first-order logic operators, respectively ∃ and ∀. However, there are still situations where all above operators will not handle to describe. Temporal logic employees two another operators dealing with key words like “until” and “unless”: (ibid.)

> pUq – *p will continuously hold from now until the moment when q starts holding.*
>
> pWq – *p will continuously hold form now on unless q will occurs in which case p will cease.*

Operators presented above refer only to future and tries to relate it with the present. There is also a way to describe past. This sort of connectives were introduced to temporal logic later, probably there was no need to reach the past. Therefore we can take an operator ○ and replace world “next” with a “previous”: (ibid., 521)

> ● p - *p is true at the previous moment in time.*

There is almost trivial relation between the operator ○ and ●, which can be shown in one short sentence p ⇔ ●○p. (ibid., 521)

Each of presented operator can be transformed into first-order logic statement. In order to make it possible we are obliged to use predicates where passed arguments are points in time i.e. natural numbers. This approach is most often called the *temporal arguments*. (Van Harmelen, Lifschitz & Porter 2008, 522)

$$\circ\, q \Rightarrow q(i\ +\ 1)$$

$$\Diamond\, r \Rightarrow (\exists j)\ (j\ \geq\ i) \wedge r(j)$$

$$\Box s \Rightarrow (\forall k)\ (k\ \geq\ i) \Rightarrow s(k)$$

3.2.3 Temporal Reasoning

Both structures and language has been discussed at the most general level. Getting experience with propositional logic, similar techniques can be applied into temporal logic having as an outcome temporal reasoning. (Van Harmelen, Lifschitz & Porter 2008, 528)

One of possible theory, which can be built on temporal logic and its structures, is proof systems. Most of such systems were based on already existing systems like provers designed for first-order logic. (Van Harmelen, Lifschitz & Porter 2008, 529)

Recently the most used method for automated reasoning was resolution based approach. The method was extended for temporal reasoning. (Van Harmelen, Lifschitz & Porter 2008, 529-530)

3.3 Spatial Knowledge and Reasoning

After AI research has begun and the idea about agent has propagated in scientific environment, most of scientists started to think how space can be perceived by the agent. This is how *qualitative spatial reasoning (QSR)* has been developed. This kind a reasoning is based on a knowledge learnt by perceiving three-dimensional space. If the agent has to use its actuators e.g. wheels, it shall know the direction and the distance according to which it will move. The first thing the agent has to come up is a complexity of 3D space measurements and location in space. (Van Harmelen, Lifschitz & Porter 2008, 551-552)

A result of the research is a knowledge that it is not possible to create purely qualitative reasoners about spatial mechanisms. The point of qualitative spatial reasoning is to create calculus, which allows the agent to reason without referring to quantitative techniques like computer vison systems and graphics. Thus, the entity does not have to know the exact shape, orientation, distance, size or topology. With qualitative reasoning and knowledge, these properties can be evaluated in a human-like manner. Human is good example of the qualitative reasoner. There is no need to measure the distance from a wall, but still it can be evaluated. This information provides with the picture of the space around. Many systems have been discovered and developed by being motivated by QSR. Robotic navigation, Geographic Information System - GIS, design or common-sense reasoning about physical systems. (Van Harmelen, Lifschitz & Porter 2008, 553-554)

3.3.1 Spatial Ontology and Relations

According to geometrical theory, all objects are represented by so-called primitives, pure spatial entities such as points, lines, planes or regions. In mathematical theory, line and plane is set of points in a special relation. Region is also a set of points, which can somehow be limited to represent a square or circle in either 3D or 2D figures. However, QSR does not use points and lines as primitives, there rather strong motivation to take regions as primitives and perform reasoning in terms of those objects. Spatial objects are represented as a set of regions instead of set of points. This approach makes sense for qualitative spatial reasoning. There are still not solved issues and ontological questions about the nature of the space i.e. what can be universal spatial entity or should null regions be excluded. (Van Harmelen, Lifschitz & Porter 2008, 556)

Qualitative spatial reasoning defines the fundamental entities and treats them as primitives. Although having just plain objects does not make the theory useful. Relating objects between each other gives powerful tool to manipulate them. Relation is defined as set of tuples of the same arity. Each element x_i of a tuple is member of considered domain X_i. In general, relation is defined as a cross-product of m domains: (Enderton 1977, 39-42)

$$R \subseteq X_1 \times \ldots \times X_m$$

$$X_1 \times \ldots \times X_m = \{(x_1, x_2, \ldots, x_m) |\, x_1 \in X_1 \wedge x_2 \in X_2 \wedge \ldots \wedge x_m \in X_m\}$$

Most often binary relations are used for which in previous case $m = 2$. We can define some operation on relations like union, intersection and other, thus creating algebra of relations. (ibid., 39-42) In qualitative representation and reasoning finite relations such as jointly exhaustive and pairwise disjoint, in short, JEPDs are usually used. They are said to be atomic or basic relations. (Van Harmelen, Lifschitz & Porter 2008, 556)

3.3.2 Mereology and Mereotopology

Knowing what is relation and having fundamental experience with operations, one can apply it in qualitative spatial reasoning. The definition of relation is influential especially in a science named mereology. (Van Harmelen, Lifschitz & Porter 2008, 557)

Mereology is concerned with the theory of parthood, deriving from the Greek μερος (part), and it forms a fundamental aspect of spatial representation, with practical applications in many fields. (ibid., 557)

There are plenty of mereological theories and choosing the right one depends mostly on what properties one wants to associate with. In qualitative spatial reasoning, the most common theory is *minimal extensional mereology (MEM)*. The theory consists of four axioms where the most important part of them is basic relation, symbolically presented as PP, which means *proper part.* (Simons 2000, 25-30)

1. Any axiom set sufficient for first-order predicate calculus with identity.
2. $(\forall x, y)[PP(x, y) \Rightarrow \neg\, PP(y, x)]$
3. $(\forall x, y, z)\{[PP(x, y) \wedge PP(y, z)] \Rightarrow PP(x, z)\}$

There is also relation O defined as overlapping, which states that two elements have common part: (Simons 2000, 25-30)

4. $(\forall x, y)[PP(x, y) \Rightarrow (\exists z)(PP(z, y) \wedge \neg O(z, x))]$

Who is familiar with mathematical fundamentals can easily notice that axiom number 2 and 3 define so called partial order in a set of elements. The last axiom is referring to *Weak Supplementation Principle (WSP)*. (Calosi 2016, 3)

Mereotopology is even more important than mereology. It combines both mereology and topology into one body, which is concerned with the relations between parts and boundaries of parts. Topology component is concerned with the concept of wholeness or connection. There is not straightforward way to merge those two fields. Mereology can be made more general by adding a topological primitive such as $S_C(x)$ = x is self-connected. This predicate states that an element is one piece instead of many. That was one approach to integrate topology and mereology. Another one is to think about topology as the main theory and treat mereology as a subtheory. This is good way to make one unified theory, however, there are difficulties with that. (Van Harmelen, Lifschitz & Porter 2008, 558)

There are axioms that gives some interpretations to mereotopology primitives. It can be assumed that P is parthood and C is connection. Therefore, the axiom sounds: (Achille 1998, 2)

Table 3 -Set of axioms of mereotopology presented in textual format. (Achille 1998, 2)

P-reflexivity	*Everything is a part of itself.*
P-antisymmetry	*Two distinct things cannot be part of each other.*
P-transitivity	*Any part of a part of a thing is itself a part of that thing.*
C-reflexivity	*Everything is connected to itself.*
C-symmetry	*If a thing is connected to a second thing, the second is connected to the first.*
Monotonicity	*Everything is connected to anything to which its parts are connected.*
P-extensionality	*No two distinct things have the same proper parts.*
P-fusion	*For any number of things there is a smallest thing of which they are all part.*
External contact	*Everything is connected with its mereological complement.*

4 NEURAL NETWORK THEORY

4.1 Foundations of Neural Network Theory

The fundamental element in an artificial neural network is a unit called neuron. In the computation theory, neuron does not reflect a real, physical neuron that can be found in a brain, it is rather conceptually similar. The whole network of connected neurons performs actual computation. The advantage of the network is that it enables *parallel distributed processing (PDP)*. (Krose & Van der Smagt 1996, 15)

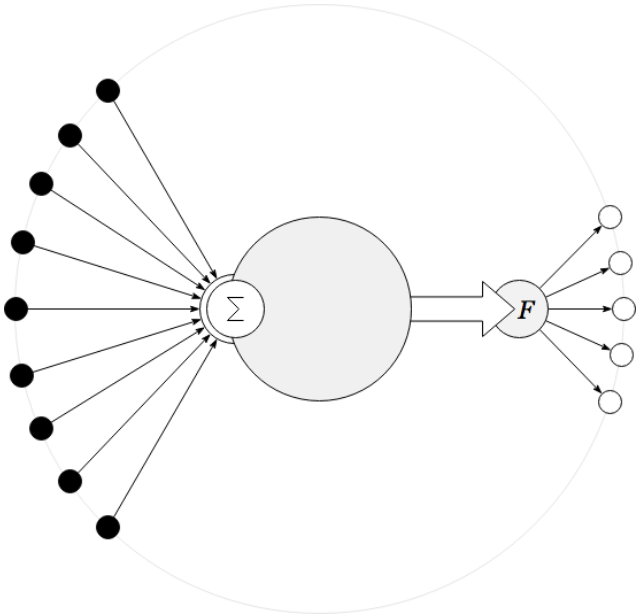


Figure 5 -The concept of the processing unit. Black filled circles represent an input of the neuron, white circles are an output. (Krose & Van der Smagt 1996, 16)

The unit takes input signals from other units or external sources and uses that information to compute an output signal. Other units are doing the same and this way signals are propagated along the network. Some operations can be done in parallel, which can save plenty of time in huge networks. There are three distinguishable types of unit; the first are *input units* that take an input signal from external source, i.e. outside the network. The second type are *hidden units*, which exist inside the network and take input signals from input units. The last type are *output units* which take input signals from hidden units and produce an output of the whole network. The output of these units is the result. (Krose & Van der Smagt 1996, 16)

Computation inside the units can be done either *synchronously* or *asynchronously*. The synchronous computation means that all units update their state of activation simultaneously – at the same time. The asynchronous computation updates the state of activation of neurons at

a time t unless the probability of updating is higher than the fixed probability assigned to the unit. (ibid., 16)

The state of activation y_k is the output of the unit. To fully understand the underlying idea is to connect units and create the network. Connections between units are defined by weights w_{jk} which determine how unit j affects the unit k. Propagation rule defines a *total input* s_k of the unit k. In general the total input $s_k(t)$ is a time function because it changes over time. The same occurs with the weight of connection between neurons $w_{jk}(t)$ and state of activation $y_k(t)$. Therefore, the input of unit k at a time t is defined as a sum of weighted activations of all units connected to neuron k. The sum can be somehow manipulated with so called bias or offset: (ibid., 16)

$$s_k(t) = \sum_j w_{jk}(t) y_k(t) + \theta_k(t)$$

It must be pointed out that there is distinction in naming the contribution depending on the sign of the weight w_{jk}. A negative value of the w_{jk} is considered as an *inhibition* and a positive values is considered as an *excitation*. The propagation rule presented above is most often used in practice; however, there are many other propositions. Figure 6. visualize the propagation rule; input units send signals to adjacent layer, where total input is obtain by aggregating products of each connections. (ibid., 16)

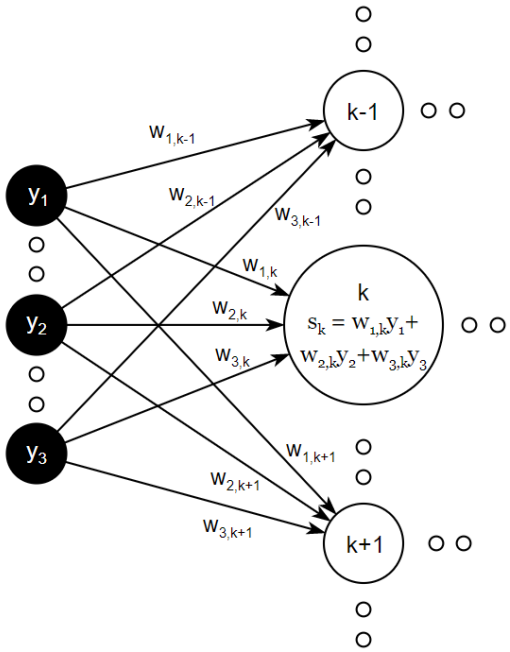


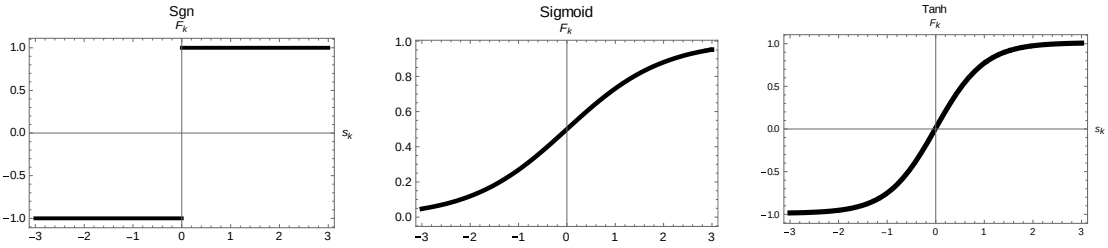
Figure 6 - The visualization of the propagation rule. Total input is the sum of weighted inputs received from input units.

The state of activation y_k is constructed by applying *activation function* F_k to the total input $s_k(t)$ and current state $y_k(t)$. Activation function determines the output of the unit: (ibid., 16)

$$y_k(t+1) = F_k(y_k(t), s_k(t))$$

Usually the activation function is required to be a nondecreasing function of only total input, which obviously depends on states of activation of other units. (Krose & Van der Smagt 1996, 17)

$$y_k(t+1) = F_k(s_k(t))$$



$F_k(s_k) = sgn(s_k)$	$F_k(s_k) = \frac{1}{1+e^{-s_k}}$	$F_k(s_k) = tanh(s_k)$
Sign Function	Sigmoid Function	Tanh Function

Figure 7 - Examples of activation function. (Krose & Van der Smagt 1996, 17)

4.1.1 Network Topologies

A neural network topology describes how neurons are organized in the network. The topology has a significant impact on the functionality and computation performance. A definition of the neural network topology consists of so-called *neural framework* and *interconnection structure*. (Krose & Van der Smagt 1996, 17)

Most of the neural networks are organized into layers; however, there are exceptions where neurons cannot be explicitly layered. In these cases, it is usually considered that the network has one layer. A good example here is Hopfield Network or Markov Chain presented in Figure 8. (Tchircoff 2017)

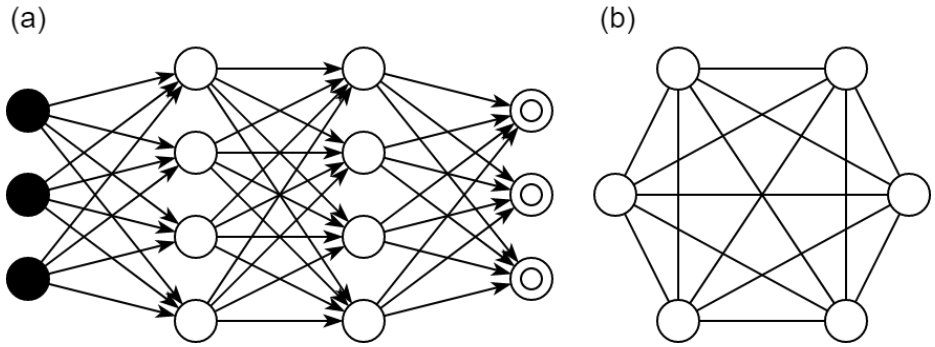


Figure 8 - Schemas of (a) multi-layer and (b) one layer network.

The neural framework consists of a *number of neural layers* L and the *number of neurons per layer* N_l. Each layer can have a different number of neurons inside. There are also cases where

neurons are grouped together in the layer but without any order. These groups are called *clusters* or *slabs*. If a network has several layers, then it is simply named *deep network*. There are three types of layers respectively to the types of neurons they consist of. *Input layer* is composed of input neurons, *hidden layer* built from hidden neurons and the last *output layer* is made of only output units. Figure 9. shows following layers. There are many networks that do not have any kind of layers, e.g. Hopfield Network or Boltzmann Machine does not have output layer and hidden layer. (Fiesler 1996, 2)

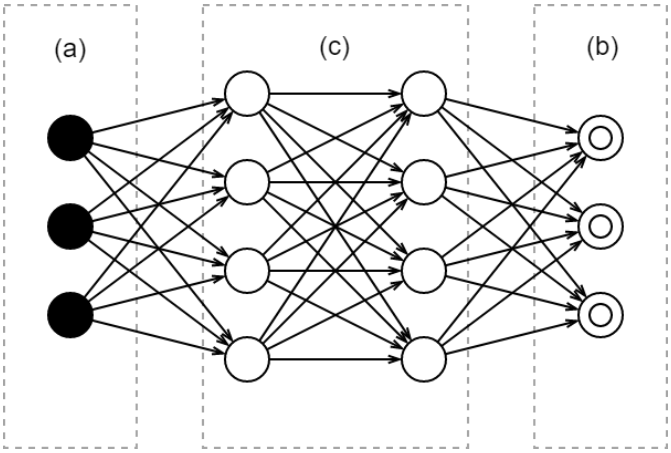


Figure 9 - Graphical visualization of (a) input, (b) output and (c) hidden layers.

The Figure 10. presents interconnection structure, which defines how units are connected to each other. There are *interlayer connections* where neurons in one layer are connected to neurons in adjacent layer. *Intralayer connection* is when neurons are connected to others in the same layer. A special case of this connection is *self-connection* where a neuron is connected with itself. The last kind is *super-layer connection* where neurons in a given layer are connected with neurons in distinct layers but not the adjacent layer. (Fiesler 1996, 3-4)

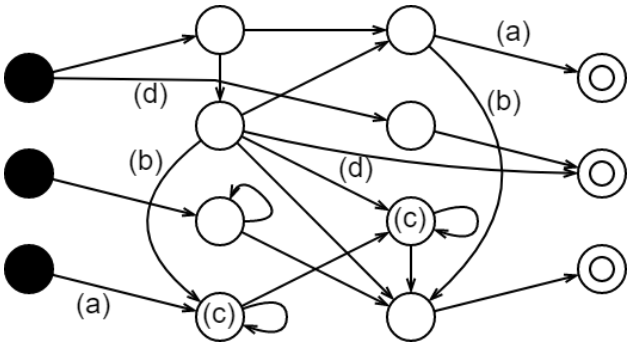


Figure 10 - The interconnection structures (a) Interlayer connection, (b) Intralayer connection, (c) self-connection and (d) super-layer connection. (Fiesler, 1996, 4.)

The neural network can be broken down by the way how data propagates. *Feed-forward Neural Network (FNN)* is a network where signal flows from input, directly through many layers to the output. There is no feedback connection present in the network. *Recurrent Neural Network (RNN)* is the network where definitely feedback connections plays a major role. In these kind a networks, the activation values for neurons are expected to be stabilized while the network is evolving in following iterations. The Figure 11. shows the idea and example of RNN and FNN. (Krose & Van der Smagt 1996, 17-18)

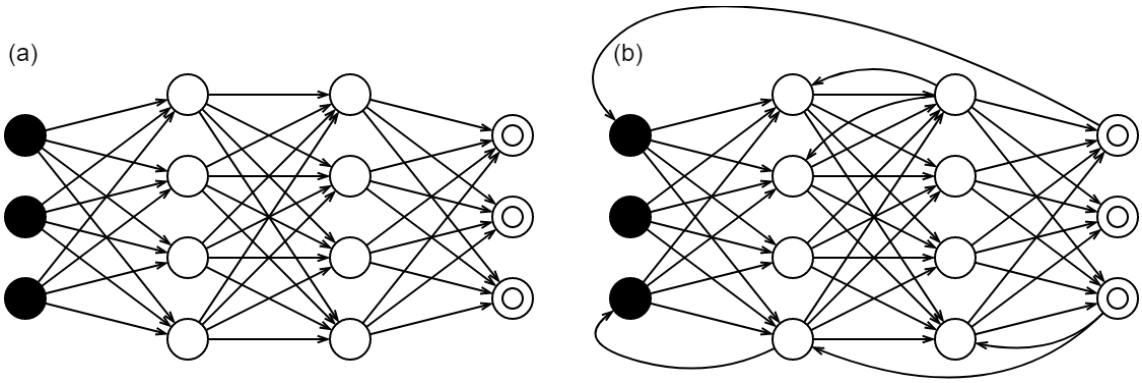


Figure 11 - Representation of (a) Feed-forward Neuron Network and (b) Recurrent Neural Network. (Krose & Van der Smagt 1996, 17-18)

4.1.2 Training Neural Networks

The neural network without meaningful data is simply useless. No one can set connection weights in such a manner that network will work properly for specified data domain. Therefore, there must be some *training method*, which will *learn* the network. There are two ways, either teaching network by giving pre-prepared data as *patterns* and change the connection weights by applying some *learning rule*. Other way is to do it explicitly using *a priori knowledge*. (Krose & Van der Smagt 1996, 18)

After all, there two distinct types of learning. *Supervised learning* is realized by giving input-output pairs delivered by external teacher. *Unsupervised learning* is based on the paradigm where the system has to examine statistically salient features of the input data. The system has to develop its own representation of an input. (ibid., 18)

Learning neural network means applying changes to the connection weights in a manner that reflects input data. Therefore, there must be some *modification rule* according to which changes can be made. One of the most famous is *Hebbian learning rule*. The underlying concept is powered by the fact that two connected neurons activated at the same time should strengthen their connection. Mathematically, when neuron k sends the output signal to the unit j , then the weight can be modified in the following way: (ibid., 18.)

$$\Delta w_{jk} = \gamma y_j y_k$$

Where coefficient γ is called *learning rate*. Another popular learning rule is *Widrow-Hoff rule*, most often named *delta rule*. The idea is to provide the desired activation d_k for a particular neuron instead of just taking activation of neuron k. (ibid., 18)

$$\Delta w_{jk} = \gamma y_j (d_k - y_k)$$

The learning method easiest to understand is *Perceptron rule*. This rule is exactly the same as Hebbian except the situations if network the responds correctly, then the connection weights are no updated. (ibid., 18)

4.1.3 Perceptron and Adaline

The Perceptron is the most basic neural network consisting of one or more artificial neurons. The network implements the learning rule, purely supervised, with the purpose to classify input vectors to one of two classes. Thus, Perceptron can be used to linear classification. (Sompolinsky 2013)

The Adaline or *adaptive linear element* was developed by Widrow and his student Hoff. (Widrow 1960, 1-10) The difference between Perceptron and Adeline is that in the learning weights are updated according to output of the system. (Krose & Van der Smagt 1996, 27-28)

4.2 Back Propagation Learning Rule

If feed-forward network has only one input layer and output units, then the learning process is easy and does not require any other methods. However, matters get complicated when more layers are added to the network. The problem concerns updating weights in hidden layers. In 1986 Williams, Hinton and Rumelhart proposed a solution for the problem. The idea is to calculate the error for output units and propagate this value back to the network; this method is named *back-propagation learning rule (BP)*. (Krose & Van der Smagt 1996, 33)

Linear classification consists of finding so-called *linear discriminant function (LDF)*. The purpose of the linear function is to split elements with different features into two or more classes. For two dimensional input data, vectors like $[x, y]$, this function is straight line. For three-dimensional data, 3D vectors, the line becomes plane. The plane cuts space into two sections. Elements located in first section are concerned to be classified to certain class. The elements located in the second section are concerned to be in a different class. The method can be generalized for higher dimensions, and then hyper-plane will classify objects. (Krose & Van der Smagt 1996, 33)

4.2.1 Generalization of Delta Rule

As noted in the previous chapters, the output of the unit is the result of activation function for total input for the unit. Before showing all necessary equations for computing updates for weights, it must be clear that feed-forward neural network is used with N_i input units, with a few but in general not specified number of hidden layers, and with N_o number of output units. This time, it is added that the neuron network computes the outputs y_k^p for many patters. Thus, the output of the unit can be presented (Krose & Van der Smagt 1996 33-35)

$$y_k^p = F_k(s_k^p)$$

Delta rule uses error function to appropriately update weights. For one output unit we measure can be defined how the actual output y_o^p of this unit differs from the desired output d_o^p. The error is calculated according to following pattern (ibid., 33-35)

$$E_o^p = \frac{1}{2}(d_o^p - y_o^p)^2$$

So for one pattern there is (Krose & Van der Smagt 1996 33-35)

$$E^p = \frac{1}{2}\sum_{o=1}^{N_o}(d_o^p - y_o^p)^2$$

Further, error function E is a sum errors for all patterns, thus, summed squared error is obtained: (ibid., 33-35)

$$E = \sum_p E^p = \frac{1}{2}\sum_p\sum_{o=1}^{N_o}(d_o^p - y_o^p)^2$$

The major idea is to change the weights proportionally to the negative derivative of the error with respect to each weight. The goal is to minimize the error with so-called *gradient descent* method: (ibid., 33-35)

$$\Delta_p w_{jk} = -\gamma\frac{\partial E^p}{\partial w_{jk}}$$

To develop the formula a *chain rule* for derivations can be used: (ibid., 33-35)

$$\frac{\partial E^p}{\partial w_{jk}} = \frac{\partial E^p}{\partial s_k^p}\frac{\partial s_k^p}{\partial w_{jk}}$$

Furthermore, it is known that $s_k^p = \sum_j w_{jk}y_j^p$ therefore, it can be said: (ibid., 33-35)

$$\frac{\partial s_k^p}{\partial w_{jk}} = y_j^p$$

If defined: (ibid., 33-35)

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p}$$

Then the update delta for weight for connection between unit j and k is shortly: (ibid., 33-35)

$$\Delta_p w_{jk} = \gamma\delta_k^p y_j^p$$

Although this is not the end of transformation of the formula. To compute above each component of the formula needs to be computed; thus, once again chain rule has to be applied: (ibid., 33-35)

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} = -\frac{\partial E^p}{\partial y_k^p}\frac{\partial y_k^p}{\partial s_k^p}$$

The second factor is finally the derivation of the activation function with respect to the total input, provided it is differentiable: (ibid., 33-35)

$$\frac{\partial y_k^p}{\partial s_k^p} = F_k{}'(s_k^p)$$

General case has been considered, i.e. for neuron k. Next is taken the output unit indexed within o, then $k = o$, this is something that can be inferred from definition of E^p: (Krose & Van der Smagt 1996, 33-35)

$$\frac{\partial E^p}{\partial y_o^p} = -(d_o^p - y_o^p)$$

Therefore the component δ_k^p for output unit: (ibid., 33-35)

$$\delta_o^p = (d_o^p - y_o^p) F_o{}'(s_o^p)$$

For hidden units after applying some transformation it can be derived that: (ibid., 33-35)

$$\delta_h^p = F_h{}'(s_h^p) \sum_{o=1}^{N_o} \delta_o^p w_{ho}$$

The last equation provides clues how to compute delta for the hidden units. (ibid., 33-35)

4.2.2 Applications

Most of the modern neural networks use back-propagation rule to adjust the weights in hidden layers. Below are some examples: (Belanche 2011)

- NETTalk – project to which one of the goals was to learn neural network to pronounce English words.
- Zipcode Recognition – the projects are concerned with recognizing numbers form pictures.
- ALVINN – Autonomous Land Vehicle in a Neural Network – project which was able to programmatically control steering of a car on a winding road.
- Face Recognition – Recognizing human face. (Belanche 2011)
- Darknet, YOLO - implementation of the neural network for general-purpose object detection. (Redmon, Divvala, Girshick & Farhadi 2015)

4.3 Recurrent Neural Networks

The Recurrent Neural Network is the network where feedback connections between units are present. This time, the neural network has cycles, so the situation looks slightly, not to say totally different. The output can be connected with input units, hidden with input units, hidden unit with itself and so forth. The capabilities of the network will not change if recurrent connections are employed. However, the model can benefit from network size or decreased complexity. Signal in RNN might be propagated over and over again through the network, and this process very likely may go on forever. However, such cases can be pointless as to be

useless. There are networks where the signal propagation is repeated several times until the network reaches so called *stable point* or *attractor*. Important fact about these kind networks is that they are able to remember previous state of the input signals. (Krose & Van der Smagt 1996, 47-50)

Plenty of different kind of RNNs have been developed. Among these networks, there are those worth to mention. The network where the output of hidden units is passed back as an input to input units is named *Elman Network*. If just connections are changed from output to hidden units then the *Jordan Network* is obtained. (Hrolenok 2009)

4.4 Hopfield Network

The Hopfield Network is an example of a recurrent neural network. Each unit is connected to each other, thus creating *complete graph*. Each neuron is an input and output at the same time. There are two versions of Hopfield network. The difference between them is that one represents the states of activation of neurons using only two values – *binary neurons with discrete time*. The second type is called *graded neurons with continuous time*, where states of activations are represented as *monotone increasing function*, bounded below and bounded above as described in Figure 12. (Hopfield 2007)

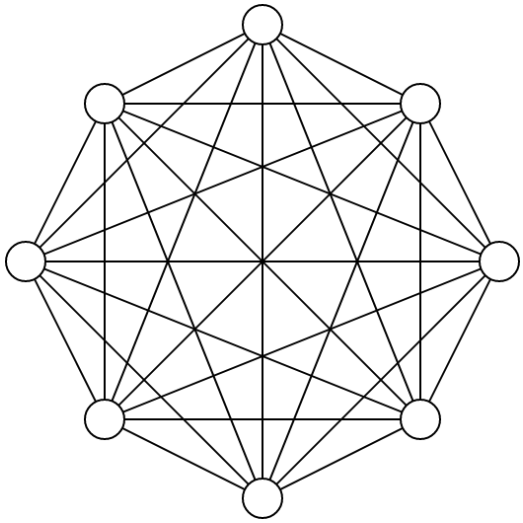


Figure 12 - The model of the Hopfield Neural Network. This is also an example of a one-layer network. (Krose & Van der Smagt 1996, 51)

4.4.1 Binary Neurons

The network does not differ much from other networks. There is only one connection between each pair of units $w_{jk} = w_{kj}$ and there are no self-connected units inside the network (w_{kk} does not exists or is set to null value). The total input s_k for unit k is represented traditionally: (Krose & Van der Smagt 1996, 50-51; Hopfield 2007)

$$s_k(t+1) = \sum_{j \neq k} w_{jk}(t) y_k(t) + \theta_k$$

In addition, the threshold function to compute the output for the unit k is as simple as possible: (Hopfield 2007)

$$y_k = \begin{cases} +1 & if\ s_k(t+1) > \theta \\ -1 & otherwise \end{cases}$$

There is benefit to choose the set $\{-1, +1\}$ instead of $\{0, 1\}$ for representing activity of neurons. First, number $+1$ and -1 are located symmetrically in relation to 0. Secondly, if some learned pattern is stable, then the negation of the pattern is stable as well. (Krose & Van der Smagt 1996, 52)

It is said that neuron k is stable at time t when the equation holds:

$$y_k(t) = sgn(s_k(t-1))$$

Thus, the neural network is named stable when all neurons in the network are in the stable state. (Krose & Van der Smagt 1996, 51)

4.4.2 Lyapunov Function

The Hopfield neural network has symmetric connections, and there are no self-connections. Therefore, the network has so-called Lyapunov function. (Hopfield 2007)

Vector field is defined as $f: X \to X$, $X \subseteq R^n$ where f is a differentiable function. *Lyapunov function* for f on U is a differentiable function $V: U \to R$ such that domain is an open subset $U \subseteq X$, and below statement holds (Lyapunov Function 2011):

$$(\forall x \in U)\ (\nabla V(x)^T f(x) \leq 0)$$

Mentioned set X is basically a set of n-dimensional vectors. The operator ∇ is called nabla and in the above expression uses for computing a gradient of a scalar field V. The gradient in R^n system is defined (Gibbs & Wilson 1960):

$$\nabla f = \left(\frac{\partial f}{\partial x_1} + \cdots + \frac{\partial f}{\partial x_n}\right) = \sum_{i=1}^{n} \overrightarrow{e_i} \frac{\partial f}{\partial x_i}$$

In neural network theory, the Lyapunov function is used to compute property associated to the network named energy. Thus, the Lyapunov function serves as energy function. After applying appropriate symbols to function, the energy looks like: (Krose & Van der Smagt 1996, 50-51; Hopfield 2007)

$$E = -\frac{1}{2}\sum_j \sum_k w_{jk} y_j y_k - \sum_k \theta_k y_k$$

5 Associative Neural Memories

5.1 Idea of Associative Memories

Learning is the process of gathering knowledge about an environment. The knowledge is stored in the base that is memory. Thanks to memory, agents can acquire and retrieve numerous of different clues about the world. This information are critically important in order to behave appropriately upon the encountered context.

Associative learning consists of learning the properties of received stimulus and also memorizing relations between stimuli and reactions. A *non-associative learning* is only acquiring the properties received by stimuli. The fully operating memory in dynamic environment is made of two parts. *Short Term Memory (STM)* can acquire knowledge and store it for a few minutes. This memory makes the agent understand the sequence of received information in a short term. The second is *Long Term Memory (LTM)* which makes the previously acquired knowledge into STM permanent. (Yeruva, Murty & Prasad 2010, 1)

The memories should be design to be *associative,* i.e. stored pieces of information should be related to other pieces. Knowledge for the memory is a set of patterns. Most of associative memories have a powerful advantage called *content-addressability*, i.e. the property of the memory which can retrieve knowledge by giving a *clue* (part of an entire pattern or noisy pattern). In a computer, a logical path has to be given to the data that is wanted to be acquired; the logical path is translated into physical, and then the data can obtained. (Yeruva, Murty & Prasad 2010, 2)

The memory is *hetero-associative* if it is able to learn two patterns from different spaces, two different patterns that are associated with each other. The memory is *auto-associative* if it associates the pattern with itself, which can be incomplete or partly distorted. (Oh, Hui & Żak 2005, 3-4)

5.2 Simple Associative Memory Models

5.2.1 Simple Associative Memory

The Simple Associative Memory (SAM) is also called *correlation memory*. It can associate input vector $x^k \in R^n$ with output vector $y^k \in R^n$ with a simple matrix multiplication: (Yeruva, Murty & Prasad 2010, 2)

$$y^k = Wx^k$$

The matrix W has dimensions $L \times N$ and represents associations between input and output vectors. Because the retrieval of the patterns is based on matrix multiplication, this model is classified to be *Linear Associative Memory (LAM).* In general, the correlation memory can hold m associations i.e. pairs (x^k, y^k). In this case, matrix W is given by: (Yeruva, Murty & Prasad 2010, 2)

$$W = \sum_{k=1}^{m} y^k (x^k)^T$$

5.2.2 Simple Nonlinear Associative Memory

Simple Nonlinear Associative Memory has been constructed by using Linear Associative Memory and applying nonlinear function F for relaxing some of the constrains in the previous model. (Yeruva, Murty & Prasad 2010, 3)

$$y^k = F(Wx^k)$$

The x^k can be normalized when stored in W matrix by pattern: (ibid.)

$$W = \frac{1}{n} \sum_{k=1}^{m} y^k (x^k)^T$$

Then, there is input x^hmatrix, and the retrieval looks like this: (ibid.)

$$\widetilde{y^h} = F(y^h + \Delta^h) = F\left(y^h + \frac{1}{n} \sum_{k=1}^{m} y^k (x^k)^T x^h\right)$$

5.2.3 Optimal Linear Associative Memory

The Linear Associative Memory seems to be not optimal. Making a set of input vectors $\{x^k \mid k = 1, \dots, m\}$ linearly independent i.e. orthogonal to each other, much better pattern learning can be designed. Thus, there is *Optimal Linear Associative Memory (OLAM)*. If vectors x^k are linearly independent then below equation can always be solved: (ibid.)

$$Y = WX$$

Furthermore, if matrix X is square matrix, than the solution can be written: (ibid.)

$$YX^{-1} = W$$

Of course, if inverse X^{-1} exists, this is true if set of x^k is linearly independent. (ibid.)

5.3 Dynamic Associative Memory Models

The association in the Linear Associative Memory can be improved by introducing structures that are more sophisticated. This improvement consists of employing dynamic architectures of associative memories. The Dynamic Associative Memory (DAM) Models are based on so-called *attractors*. The patterns, e.g. images or binary matrices, are stored in stable states of the dynamic memories, and the state changes of the system are under way to closest attractor. (Reznik & Dziuba 2009, 1)

The most important advantage of the dynamic memories is that they do not need full vector as an input to retrieve memorized pattern. Memory which memorized m pattern vectors $x^k$is considered next. Some noisy vector $\hat{x}^k$ can be taken where only 60% of information included inside the pattern are the same as in the original pattern x^k which has been stored in the memory. Giving the noisy vector as an input, the associative memory should retrieve a pattern that is much closer to the original one. The process can be repeated by applying retrieved patterns as the input, again and again, until the system will give the original pattern as an output. Such a behaviour is rather not guaranteed. (Yeruva, Murty & Prasad 2010, 3)

5.3.1 Hopfield Network

Hopfield Network is one of the dynamic memory systems proposed by physicist Dr John J. Hopfield at the California Institute of Technology in 1982. (Hopfield 2007) The network can retrieve patterns by stabilizing the states of all units inside. Patterns are stored in the attractors, in a distributed neural network, and stabilizing procedure has recurrent nature. The output is returned when the network is stable. Input pattern, usually called the clue, can be incomplete or somehow distorted, and the main feature of the network is that it can find the original pattern. This feature is known as *pattern completion* and there are numerous applications in image processing. (Yeruva, Murty & Prasad 2010, 3)

The network can be named *opened*, because it does not contain any hidden units. What is more, each unit of the network is input and output at the same time, and the whole network has only one layer. (Reznik & Dziuba 2009, 1)

5.3.2 Brain-State-in-a-Box

Brain-State-in-a-Box (BSB) is one of the models similar to Hopfield network proposed in 1977 by Silverstein, Ritz, Anderson and Jones. (Oh, Hui & Żak 2005, 7) The model is nonlinear auto-associative neural network and can be modified to handle hetero-associative aspect. The auto-associative model has one layer where all units are fully connected. The units are updated synchronously. The system may be used for pattern completion. (Yeruva, Murty & Prasad 2010, 3)

The BSB model can be represented by the equation – the update rule:

$$x^k = f(x^k + \vartheta W x^k)$$

Where W is a symmetric weight matrix with the condition: $(\forall x)((x^k)^T W x^k \geq 0)$, ϑ – feedback factor and f threshold function, usually semi-linear. (BSB Model 2018)

5.3.3 Bi-directional Associative Memory

Bi-directional Associative Memory (BAM) model has been officially presented by Bart Kosko in 1988. This model is similar to the Hopfield model; however, it also contains a second layer, which provides the ability of hetero-association. The units in the network are both input and output neurons and this depends on which layer has been chosen to be input or output. (BAM 2018)

The network stores patterns in a similar manner as Hopfield model. Thus, bi-directional associative memory has recurrent nature and pattern is retrieved; then the system reaches a stable state. The capacity of the network is around $\sim 0.2n$, where n is a number of units in each of two layers. (Yeruva, Murty & Prasad 2010, 5)

There are two types of the network. The first one is discrete BAM, where units store binary values, either 1 or -1. The second is continuous BAM, where units store continuous values of sigmoid or some hyperbolic function. (ibid.)

5.4 Hopfield Model as Associative Neural Memory

Hopfield Model was preliminary designed to be applied as the associative memory system, although the model can also be used in various optimisation problems for converting analog into digital signals. (Wang, Wu, Du & Zhang 2014, 2)

The Hopfield model is a continuous-time, continuous-state dynamic associative memory model. Information retrieval is realized as an evolution of the system state. The binary Hopfield network is a well-known model for nonlinear associative memories. This is done by mapping a fundamental memory $\vec{x}$ onto a stable point of a dynamical system. The states of the neurons can be considered as short-term memories (STMs) while the synaptic weights can be treated as long-term memories (LTMs). (ibid., 3-4)

5.4.1 Stability of Hopfield Network

The stable point or the attractor of the model is described by Lyapunov's second theorem. The Lyapunov's function is called an energy function, which can be treated as a cost function. The property is minimalized by the Hopfield network with a gradient-descent algorithm. Hopfield network is asymptotically stable when units are updated asynchronously. (Wang, Wu, Du & Zhang 2014, 2)

The stable state of network is recognized as local minimum of the energy function E. The idea of the memory is to give a distorted pattern as input and retrieve original one when the network stabilizes the state of each units. (Hopfield 2007)

5.4.2 Memory Storage and Retrieval

The knowledge is stored in the network as set of patterns. Each pattern is represented as bipolar vector $x^p = (x_1^p, x_2^p, \dots, x_n^p)$, where $x_i^p = \pm 1$, and index states that this is p^{th} pattern of N patterns. The profit of choosing value set $\{-1, 1\}$ instead of $\{0, 1\}$ has been discussed in chapter 4.4.1. The patterns can be called *fundamental memories* as well. (Wang, Wu, Du & Zhang 2014, 4)

The memory storage is realized by executing learning algorithm. One of the most famous learning algorithms is *Hebbian Rule*. The rule updates the weights in the connection matrix according to the expression: (ibid.)

$$w_{ij} = \frac{1}{n}\sum_{p=1}^{N} x_i^p x_j^p$$

The rule must be repeated for each i, j , except when $i = j$. The above can be also expressed in a more general form, where I is $n \times n$ identity matrix: (ibid.)

$$W = \frac{1}{n}\left(\sum_{p=1}^{N} x^p (x^p)^T - NI\right)$$

There is also a defined storage capability, where N_{max} refers to a maximum number of patterns which can be stored in the network. (ibid.)

$$N_{max} = \frac{n}{2\ln(n)}$$

Thus, to store 10 patterns with the dimension 100 elements each, we need 100 neurons. This way weight matrix must have 10 000 elements. The Hebbian rule can be improved. (ibid.)

The retrieval is computed by giving clue pattern, which can be incomplete or somehow distorted. The Hopfield associative memory should be able to retrieve full pattern, which has been stored previously. The procedure of retrieval for can be expressed: (Wang, Wu, Du & Zhang 2014, 5)

$$x_i(t+1) = sgn\left(\sum_{j=1}^{n} w_{ij} x_j(t) + \theta_j\right)$$

The formula represents i^{th}element of the retrieved pattern at time t. The procedure is repeated until vector x_i will not change the state in next iterations. The above expression can be written in more general, matrix form: (ibid.)

$$x(t+1) = sgn(Wx(t) + \theta)$$

This is shortly how the network reaches stable state. (ibid.)

5.5 Applications

The associative systems might be used in numerous fields, e.g. human resources management, financial, data mining, industrial, marketing or medical. (Yeruva, Murty & Prasad 2010, 7) Below are some examples of application:

- *Personnel Profiling* – the system can adjust employees to appropriate task by checking preferences and competitions.
- *Bankruptcy Prediction* – the system can recognize that given company is in potential bankrupt.
- *Prediction* – the system potentially is able to predict future values of some variables in a database.
- *Quality Control* – relying on characteristics, the system can predict the quality of some raw materials, check the quality of tires etc.
- *Sales Forecasting* – the system can give clues about future sales; the predictions are based on previous sales data.
- *Medical Diagnosis* – recognizes symptoms of diseases. (Yeruva, Murty & Prasad 2010, 7)

6 IMPLEMENTATION

6.1 Architecture and Technology

Neural networks usually are implemented rather with low level programming languages. The reason for that is numerical nature of neural computation. Therefore, to boost the performance the best option is to write C or C++ code. Plenty of already existing implementations, right next to Central Processing Unit (CPU) are involved in computational process also Graphics Processing Unit (GPU). This can entirely speed up the processing procedure, especially when networks have to consume and compute a huge amount of data. In general, the input data for neural networks can be whatever: ranging from text and vectors to voice stream and camera images. To be logical - the larger the set of data is, the more computation power is required. (Janakiram MSV 2018)

Hopfield neural network as an associative neural memory was implemented in Windows operating system environment. The most popular programming IDE has been used to write a code, debug the solution and to run the logic, Microsoft Visual Studio 2015 (VS 2015). As the programming language, C++ was chosen due to deep control over memory management and computational destination. There is no need to integrate an application with any database because all data, input or output, are read and written to files and stored locally on the filesystem.

6.1.1 UML Class Diagram

A code is appropriately decomposed into several classes, each embedded in a namespace. The class approach design has many benefits comparing to traditional functional paradigm. One of them is encapsulation, which helps to organise the code and logic behind it in a very advanced way.

The project consists of many different classes, some of them represent behaviours and another rather static structures and algorithms making use of a data stored in that structures. Unified Modelling Language (UML 2018) has been developed to graphically represent classes and relations between them.

The Figure 13 shows all classes in a project and describes the relations between them. LearningRule is a learning interface. Hebbian class implements this interface and adds other methods allowing performing the learning algorithm. This class associates Patterns object and HopfieldNetwork object. HopfieldNetwork is a class, which inherits from NeuralNetwork. NeuralNetwork creates a base for any kind of neural network by abstracting from topology. The Pattern class encapsulates all operations performed on bi-polar vectors and defines the structure to store data. The instances of the pattern are aggregated in Pattern's object, which is

designed to keep them in a standard list. Utility class provides operations such as exporting and importing bitmaps.

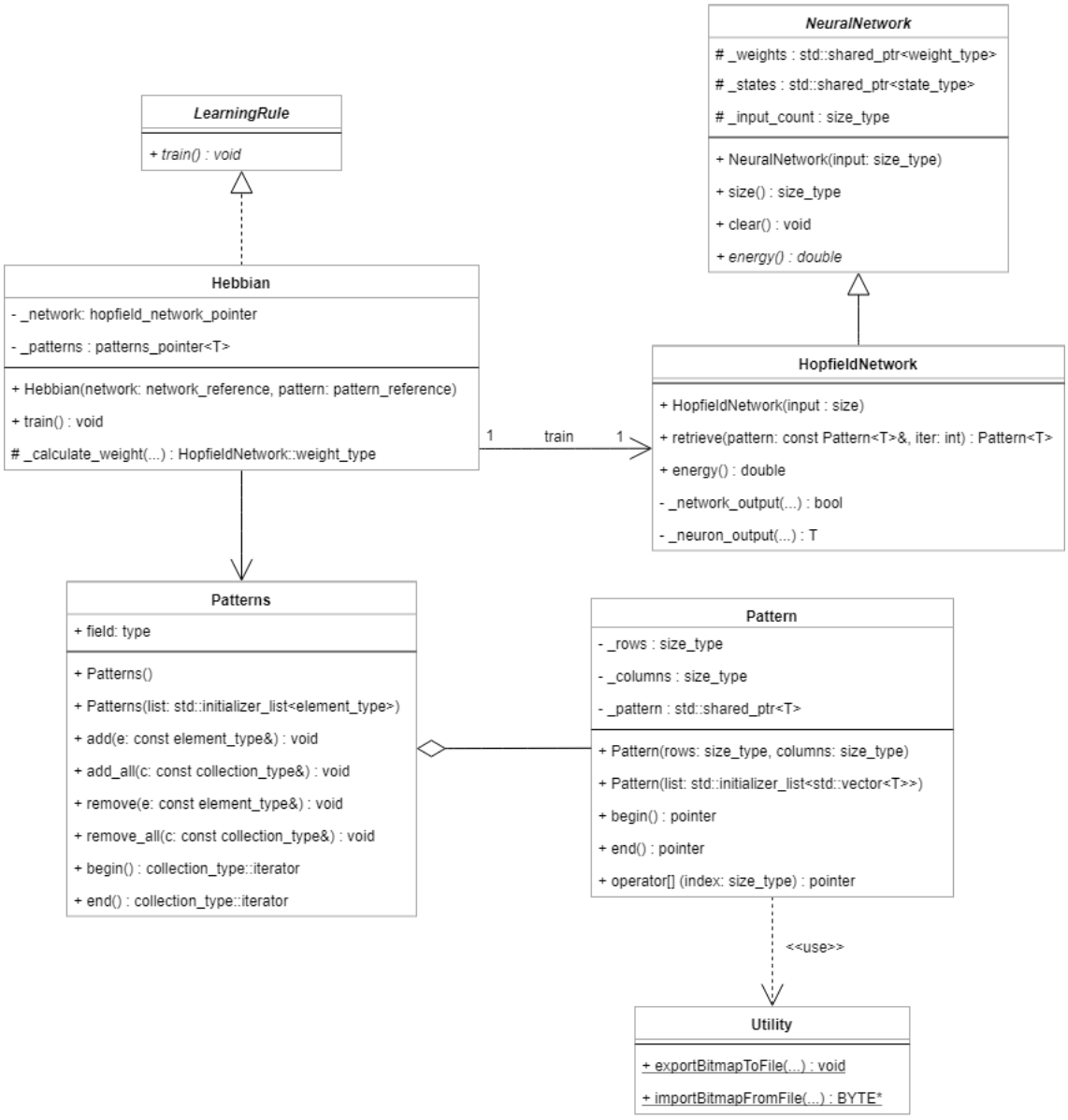


Figure 13 – The UML class diagram representing classes and relations in Hopfield Network project.

6.1.2 Data Structures

Patterns are n-dimensional bipolar vectors where each element is either -1 or 1. Vectors, by default are supported in almost every language. The pattern is usually pictured as a matrix instead of a vector. Usually patterns are presented as pixel images, where each pixel holds an information for corresponding element in bipolar vector. Images are appropriately read pixel by pixel and converted to the vector.

The Figure 14 shows how 2D matrix is flattened into 1D vector.

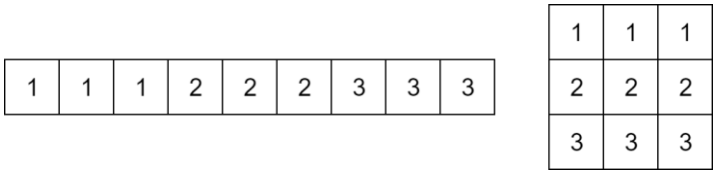


Figure 14 – Pattern converted into bipolar vector.

Hopfield network consists of neurons and connections between them. Connections are stored in matrix represented in exactly the same way as patterns. The $n \times n$ matrix has been flattened into a vector of size n^2. Another attributes of neural network are: states of whole neurons, which can be treated as the short term memory, and synaptic weights which can be treated as long term memory.

6.1.3 Programming Paradigms

The synaptic weights are represented as double precision numbers. Thus, one connection between two neurons takes 8 bytes of memory, which seems to be not much. It must be pointed out that there are thousands, if not millions of such connections in a neural network. It would be very handy if the type for synaptic weights could be easily switched between 8 and 4 bytes. However, in general, changing the type might cause an inconsistency somewhere in a code. Usually in that case there must be a different implementation for the same algorithm with only another type. To avoid this phenomenon, programmers involve generic types, templates which can abstract from the type and concentrate on the implementation of the algorithm. Sometimes, however, there are types for which the algorithm must be implemented in a different way as it was done in the generic method. Then a specification defines the actual implementation for a specific type.

Patterns are bipolar vectors, and values can either be stored in 4-byte integers or one byte characters. To easily change the type, the pattern is class template, where template parameter is the type used by the pattern. In order to use the code prepared in such a manner, the programmer has to first instantiate the template and then create an object of the instantiated class.

The C++ programming language is all about handling pointers to the memory which has been allocated somewhere in a heap (RAM). First, the memory is allocated by the “new” operator, which returns an address to the memory. The address must be assigned to the pointer. Once the memory has been allocated, then after all computation, it must be released to be reused again. The problem in a complex model is that pointers are passed to many places in the code, and it is hard to tell, in which point it should be freed. Therefore, smart pointers have been discovered to handle such unfortunate cases. Smart pointer is a wrapper that stores the raw pointer, and usually it also stores a reference counter, which informs how many references there are in the code. If the reference counter is equal to zero, then the memory is released automatically. This is how the problem has been solved.

Another technique worth mentioning is Resource Acquisition Is Instantiation (RAII). The method stands for acquiring memory during instantiation and releasing it on destruction. This technique helps programmers to write code that is more intuitive and avoid the need to know where the memory should be released. (RAII Cppreference 2018)

6.2 Neural Network

HopfieldNetwork class has been introduced to encapsulate the computation process of memory storage and retrieval. The class inherits from a base class NeuralNetwork. First, a namespace is declared and the class within it. Figure 15 shows the details about the declaration of the NeuralNetwork and HopfieldNetowrk classes.

```
namespace nn
{
     class NeuralNetwork
     {
     public:
             typedef double weight_type;
             typedef char state_type;
             typedef int size_type;

             NeuralNetwork(size_type);
             virtual ~NeuralNetwork();
             // . . .
             virtual double energy() = 0;
     protected:
             // . . .
             std::shared_ptr<weight_type> _weights;
             std::shared_ptr<state_type> _states;
             state_type _treshhold_value;
             size_type _input_count;
     };
     // . . .

     class HopfieldNetwork : public NeuralNetwork
     {
     public:
             HopfieldNetwork(size_type);
             virtual ~HopfieldNetwork();

             template<class T>
             Pattern<T> retrieve(const Pattern<T>&, int);

             virtual double energy();
     private:
             // . . .
     };
}
```

Figure 15 – The declaration of the HopfieldNetwork class.

NeuralNetwork is the base class, which defines its own types to represent synaptic weights, states of neurons and finally, the input size. At the end of class declaration, there are private fields, which store all those attributes of the neural network. Especially the line below declares shared pointer to the memory, where synaptic weights are allocated

```
std::shared_ptr<weight_type> _weights;
```

A class std::shared_ptr<T> is a class template, which stores a raw pointer to the type T. Shared pointer is one kind of a smart pointer. HopfieldNetwork class publicly inherits form NeuralNetwork base class and overwrites energy function.

6.2.1 Memory Retrieval Algorithm

Figure 15 shows two functions, the most important of which is HopfieldNetwork::retrieve(), which takes a clue pattern as an argument.

```
template<class T>
Pattern<T> HopfieldNetwork::retrieve(
                const Pattern<T>& input, int max_iterations)
{
    // . . .
    _init_index_array(index, size);

    while (1)
    {
            _shuffle_index_array(index, size);
            has_changed = _network_output<T>(index);

            ++iterations;
            if (!has_changed || iterations >= max_iterations)
                    break;
    }
    // . . .
    return result;
}
```

Figure 16 – The memory retrieval algorithm.

Figure 16 presents the algorithm, which retrieves the memory from the neural network. There is an infinite loop which repeats the steps of the algorithm. Inside the loop there is condition which can interrupt the computation. There is limit for the maximum number of iterations given as parameter for the function. The actual computation is repeated until the state of the output is not changing, i.e. attractor. The method takes the clue pattern as the input and copies this pattern to the result pattern. Therefore, initially, the output pattern is the same as input pattern in a sense that all values are copied. In each iteration, a _network_output() function is invoked which changes the result pattern. An array index is created to store indexes to neurons. The function _sfuffle_index_array() takes the array and shuffles indexes. Thanks to that, neurons are accessed randomly instead of being accessed sequentially.

6.2.2 Energy Function

The second function, HopfieldNetwork::energy() computes the current state of the network. The function has been implemented according to the mathematical pattern explained in chapter 4.4.2, which is presented in Figure 17.

```
double HopfieldNetwork::energy()
{
	double e = 0.;
	weight_type* weights_ptr = _weights.get();
	state_type* states_ptr = _states.get();

	for (size_type i = 0; i < _input_count; ++i)
		for (size_type j = 0; j < _input_count; ++j)
			e += weights_ptr[i * _input_count + j] *
				states_ptr[i] * states_ptr[j];

	e = -0.5 * e;
	return e;
}
```

Figure 17 – The implementation of Lyapunov Function; Energy Function of the neural network.

6.3 Learning Rule

Hebbian class inherits from LearningRule class. LearningRule is an interface declaring pure virtual train() method. The learning algorithm is performed on a neural network, where synaptic weights are updated according to passed patterns. The patterns are learning data set for the neural network. Therefore, the class consists of the pointer to the network and the pointer to a list of patterns.

```
namespace nn::learning
{
	class LearningRule
	{
	public:
		virtual void train() = 0;
	};
	// . . .

	template<class T>
	class Hebbian : public LearningRule
	{
	public:
		// . . .
		typedef Patterns<value_type>& patterns_reference;
		typedef HopfieldNetwork& network_reference;

		explicit Hebbian(hopfield_network_pointer, patterns_pointer<value_type>);
		explicit Hebbian(network_reference, patterns_reference);
		Hebbian(const Hebbian&);
		virtual ~Hebbian();

		// . . .
		virtual void train();
	protected:
```

```
            typename HopfieldNetwork::weight_type _calculate_weight(
                    HopfieldNetwork::size_type, HopfieldNetwork::size_type);

    private:
            hopfield_network_pointer _network;
            patterns_pointer<value_type> _patterns;
    };
}
```

Figure 18 – Declaration of the Hebbian class; the learing rule wrapper.

Figure 18 presents the declaration of the learning rule class. In this case, Hebbian learning rule has been applied. The class defines its own types for template argument, e.g. network_reference is an alias for the type HopfieldNetwork&, etc. There are two explicit constructors, one of them takes pointer as parameters, the other takes references. An explicit key word means in this context that an object cannot be instantiated implicitly, and only explicit expression must be specified to call the instantiation.

6.3.1 Bi-polar Vector Pattern

Pattern class is a data structure, where bi-polar vectors are stored. Figure 19 shows how a pattern has been declared. To accomplish the generality, the template paradigm has been used. The benefit of this approach is the template can be generated for any type. However, an implementation of the template class limits a type domain; not every type can be passed as a parameter to the template. Thus, Pattern class will work for primitive types, i.e. char, wchar, int, float, double and any aliases of the listed types it was designed for.

Another feature of Pattern class is list initialization. This can be achieved by creating a constructor of the object, which takes initializer_list object as parameter, which enables specifying instantiation of the Pattern class as presented in Figure 20.

Pattern class also declares move constructor. C++ language is known for so-called move semantics. Move constructor ensures that instead of creating a copy of an object, the memory associated with it will be moved. The mentioned semantic can be implemented by coping addresses. This provides optimization when object is passed multiple times from one stack's frame to another. The constructor must be declared as Pattern::Pattern(self&& other), where symbol && states for r-value reference. Figure 19 shows also that class has overloaded operator= and operator[]. There are many functions like begin(), end(); combinations of them with const key word. These methods enable to iterate over internal structure by using foreach statement.

```
namespace nn::learning
{
    template<class T>
    class Pattern
    {
    public:
            // . . .
    private:
            using self = Pattern<T>;
```

```
            size_type _rows, _columns;
            std::shared_ptr<value_type> _pattern;
    public:
            explicit Pattern(size_type rows, size_type columns);
            explicit Pattern(std::initializer_list<std::vector<value_type>> list);
            Pattern(const self& other);
            Pattern(self&& other);

            self& operator= (const self& other);
            pointer operator[] (size_type index);
            const_pointer operator[] (size_type index) const;

            pointer begin();
            pointer begin() const;
            const_pointer cbegin() const;
            pointer end();
            pointer end() const;
            const_pointer cend() const;
            // . . .
    };
}
```

Figure 19 – Declaration of the Pattern class.

Figure 20 presents how Pattern class can be used.

```
// List instantiation:
Pattern<int> pattern({
     { 1,1,1,1,1 },
     { 1,1,1,1,0 },
     { 1,1,1,0,0 },
     { 1,1,0,0,0 },
     { 1,0,0,0,0 },
});

// Assigning elements to pattern:
pattern[1][2] = 1;
pattern[3][0] = 0;

// Creating the copy of the pattern instance:
Pattern<int> pattern1 = pattern;
pattern1 += 5; // += is overloaded

// Iterating over the elements:
for (auto& elem : pattern)
     elem = 2 * elem – 1;
```

Figure 20 – Example of usage of the Pattern class.

Sometimes there is need to perform some arithmetic operation for each element. Standard operations can be declared out of the class declaration. Figure 21 shows the mentioned idea. Following templates make it possible to potentially add two patterns with different types.

```
template<class T, class U>
nn::learning::Pattern<T> operator+ (const nn::learning::Pattern<T>&,
     const nn::learning::Pattern<U>&);

template<class T, class U>
nn::learning::Pattern<T> operator- (const nn::learning::Pattern<T>&,
     const nn::learning::Pattern<U>&);

template<class T, class U>
nn::learning::Pattern<T> operator* (const nn::learning::Pattern<T>&, U);
```

Figure 21 – Overloading some of arithmetic operators.

Patterns can also be exported to and imported from bitmap. There are two functions presented in Figure 22.

```
template<class T>
std::shared_ptr<Pattern<T>> import_bitmap(int, std::string);

template<class T>
bool export_bitmap(const Pattern<T>&, int, std::string);
```

Figure 22 – Functions to import and export pattern to the bitmap format.

6.3.2 Hebbian Rule

Figure 18 explained the declaration of Hebbian learning rule; however, an implementation does the actual job. Hebbian rule is concerned as the most basic and obvious method, therefore it is not complicated if it comes to implementation. Figure 23 illustrates how learning has been implemented:

```
template<class T>
void Hebbian<T>::train()
{
     // . . .
     // get raw pointer:
     nn::HopfieldNetwork::weight_type* weights =
             _network->get_weights().get();

     for (auto i = 0; i < neuron_no; ++i)
     {
             for (auto j = 0; j < neuron_no; ++j)
             {
                     if (j == i) continue;
                     weights[i * neuron_no + j] = _calculate_weight(i, j);
             }
     }
}

template<class T>
typename nn::HopfieldNetwork::weight_type Hebbian<T>::_calculate_weight(
     nn::HopfieldNetwork::size_type i, nn::HopfieldNetwork::size_type j)
```

```
{
     HopfieldNetwork::weight_type w_ij = 0.0;
     Pattern<T>::size_type size = 1.0;
     for (auto& pattern : *_patterns)
     {
             Pattern<T>::value_type* ptr = pattern.get_pattern().get();
             size = pattern.rows() * pattern.columns();
             if (i >= size || j >= size) continue;
             w_ij += ptr[i] * ptr[j];
     }
     w_ij = (1. / (HopfieldNetwork::weight_type)size) * w_ij;
     return w_ij;
}
```

Figure 23 – Implementation of the Hebbian learning rule.

The method Hebbian::train() iterates over each neuron and performs Hebbian::_calculate_weight() method for each possible connection in neural network. New weight is obtained by summing products of multiplication of the appropriate pattern's elements. Before the new weight is returned, it is multiplied by factor $\frac{1}{n}$, where n is a size of pattern.

6.4 Data Representation

Patterns are stored in files in a text format, where units are presented by 0s and 1s separated by spaces. The second way is to create a bitmap, where the black pixel represents 1 and white pixel represents 0. Figure 24 presents these two methods. On the right side, the upper pattern shows the text format, however, 0s and 1s are so small that the image turns grey. For the pattern on the left side, each value is converted to a square 50 x 50 pixels, thus 10 x 10 text pattern is 500 x 500 pixels bitmap pattern.

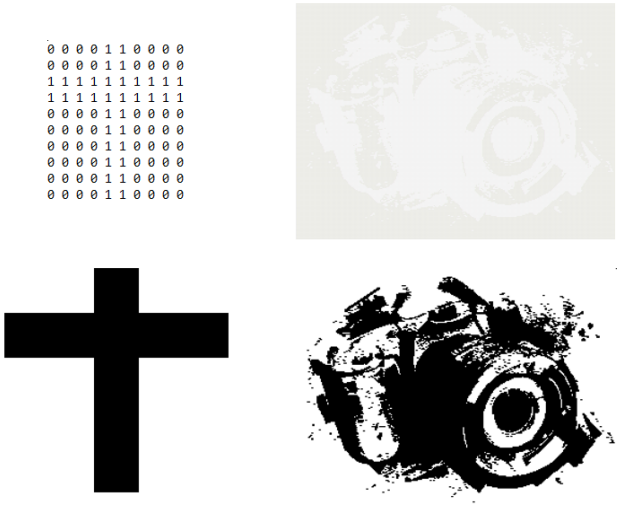


Figure 24 – Upper pictures presents text format for pattern, whereas bottom pictures shows bitmap format.

6.5 Dependencies and Version Control System

The project tries to keep the model as simple as possible. Introducing any extra complexity would dim the actual idea for implementing the associative neural memory. Therefore, the code uses only standard C++ library, which includes powerful data structures and outstanding implementation of standard algorithms.

For generating images in bitmap format, the Windows API has been used. The Utility class has an ability to export and import patterns to the bitmap file.

In a modern development a code should be kept in the so-called version control system (VCS). The system shall enable reaching the code from any location on the Earth, storing the whole history of the project and storing all changes made in the past. One of the most famous VCSs is Git, which this project also uses.

7 RESULTS

The purpose of neural network is to learn and retrieve memorized patterns. In the following subchapters, the auto-associative variant of Hopfield Neural network is tested.

7.1 Retrieval Test with Different Noise Level

Figure 25 presents a test consisting in learning neural network with many patterns. Each pattern has to be different; otherwise, the neural network during retrieval will have difficulties in distinguishing some of them; the neural network will retrieve a different output then it results from the clue pattern.

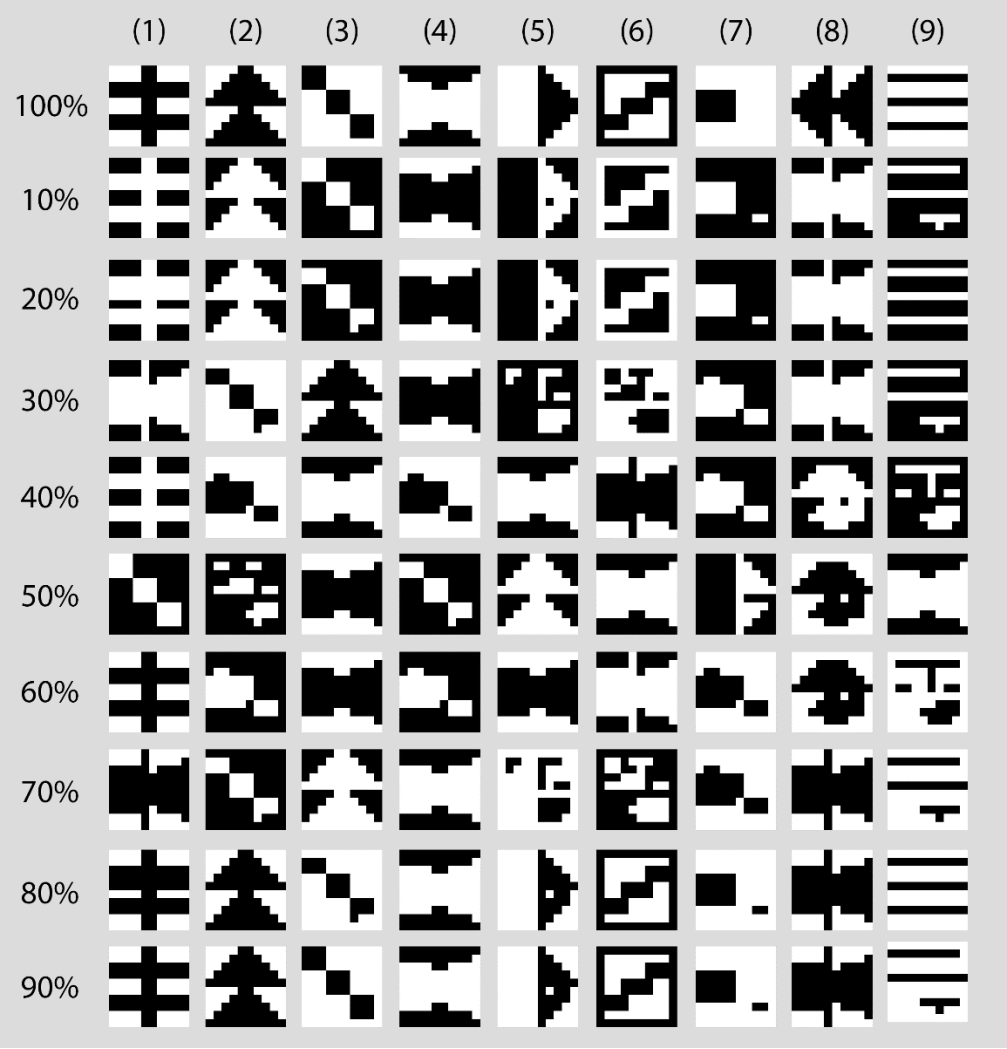


Figure 25 – Retrieval test for nine different patterns and for different level of noise.

Noise is generated automatically by negation of values at random positions in the pattern. Thus, a pattern with 40% of noise is pattern 60% similar to the original one. Therefore 40% are noisiness and 60% are likeness. Likeness of two patterns can be defined as the ratio between the number of units with the same values and number of units with different values. Of course, only units at the same positions are compared. Figure 26 presents the original pattern and the same pattern after applying auto-generated noise. If the noise is about 20%, and the pattern contains 100 values, then 20 units in this pattern will have opposite values at random positions.

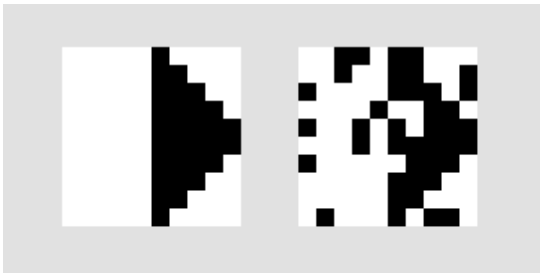


Figure 26 – Original pattern on the left side and the same pattern after applying 20% of noise on the right side.

Hopfield network takes the noisy pattern as the clue and the original pattern is expected as an output. The test is performed in this way, that nine patterns are loaded and prepared to be learned by the network. Then the actual learing process is executed. After that, one of previously memorized patterns is taken, the noise is applied at some level, and so the prepared clue is given as the input to the neural network. The output is compared with the original pattern, not the clue, and likeness is obtained. The process is repeated for different levels of noise, every 1%, and for all of the memorized patterrns. Figure 27 shows a curve which indicates what is minimal percent likeness of clue to the original pattern that the output is acceptably similar to the original pattern. It is about 70% for the clue to retrieve more then 90% similar output. Below 70%, the likeness of the output is linearly falling down. However, the figure also shows that in the range from 0% to about 30%, the situation is somehow similar to the range from 70% to 100%. This is because patterns with likeness close to 0% are negations of patterns with likeness close to 100%. Two patterns, where the second one is negation of the first one, are in a special relation; second pattern present 100% of noise for the first pattern, and these two patterns are 0% like. Figure 27. presents also a feature of Hopfield neural network, for which it is true that if a network reaches stability (attractor) for a given clue, it will also reach sa table state for the negated pattern.

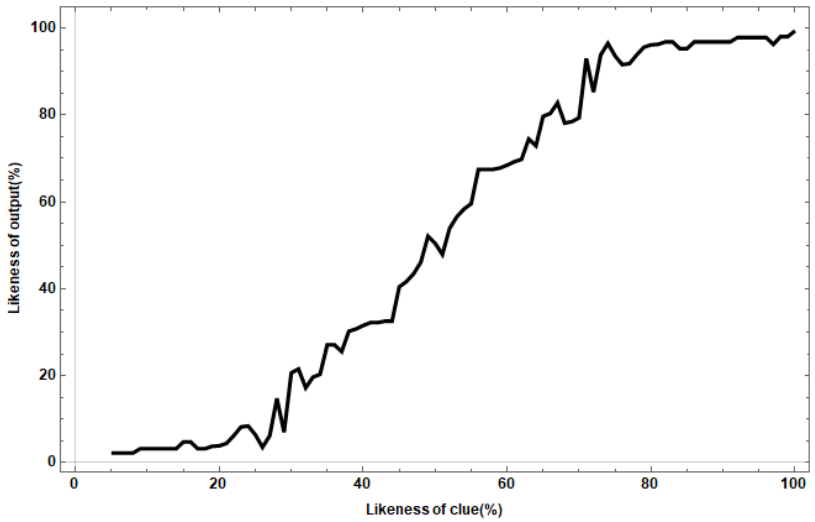


Figure 27 – Plot presenting how likeness of output depends on likeness of clue pattern.

The memory retrieval for multiple patterns with different level of noise is shown in Figure 25. One column presents the same pattern after retrieval; at least the same pattern is expected on the output. The first row consists of patterns, which were not distorted, so they are 100% alike. Below the first row, the percentage means the likeness of the clue given to the learned neural network. One can notice that for 80%, 90% and partially for 70%, the neural network gives in most cases the right pattern. The same can be noticed for the likeness up to 30%, however, this time the network gives negated results. For 50% of likeness, all received patterns are different than expected. Usually, they do not present random results but another patterns stored in the network and which do not fit to that expected one, i.e. so-called false-positives. For the pattern number eight, it does not matter how the clue is similar to the original pattern, every time a neural network gives wrong results, even for 90% of likeness. This case shows that the accuracy of memory retrieval depends on the shape of patterns. If patterns are very similar to each other, and there are only insignificant differences, then very likely the neural network will have difficulties to tell them apart.

7.2 Memory Retrieval with One Picture

Another important test is to learn network with only one pattern and try to restore the image which has an incomplete one. The figures presented below show examples of memory retrieval for three different images. The RGB image is first converted to bitmap with only two colours. The next step is learning Hopfield network with given image. After the process has been finished, a properly prepared clue image can be passed as the input to the network. The network should receive the original picture.

Figure 28 – JAMK logos and a reindeer; clues on the left side are incomplete and contain some additional pixels in the wrong place. The images on the right side show the pattern after retrieval.

7.3 Memory Retrieval with Many Pictures

Previously, the neural network memorized only one image at time, and the retrieval run only for that pattern. It is worth checking what will happen if the network learn more bigger images. Figure 29 explains this case by showing memorized pictures after the retrieval. For each pattern the same clues have been applied (from Figure 28). The image on the left side has been retrieved with 100% of accuracy. The image in the middle looks almost like the original one with some distortion, and the readability of the results is lower. Finally, the last one, the image on the right side does not look like a reindeer at all.

Another test for retrieving patterns has been performed. Figure 30 visualizes the output retrieved from the neural network. Each pattern has been retrieved for a network that has memorized a certain amount of patterns. Numbers in the figure indicate a count of patterns, which had been learnt by Hopfield network, while the retrieval process has been triggered. This shows that if more patterns are stored in the memory, a lower retrieval accuracy is gained.

Figure 29 – Memory retrieval when three pictures are memorized at the same time.

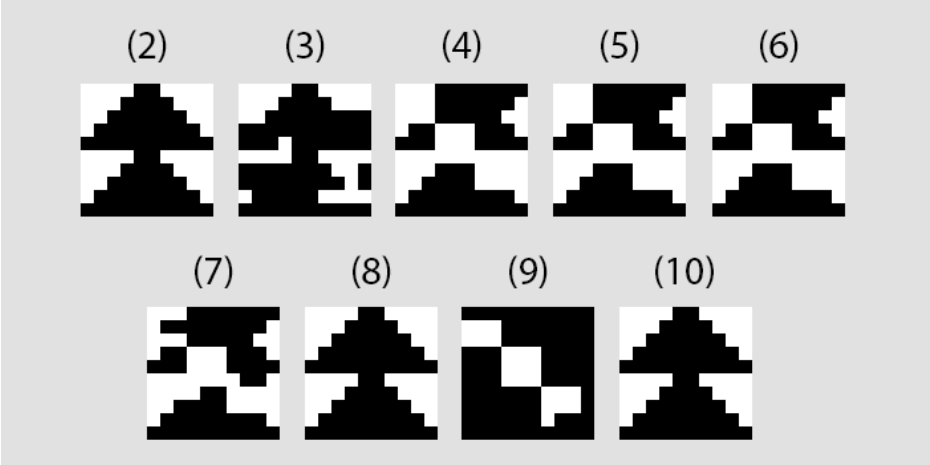


Figure 30 – Memory retrieval for a network learned each time with more patterns. Numbers tell how many different patterns has been memorized by neural network.

8 CONCLUSIONS

Hopfield Neural Network is a dynamic associative memory system and can be used for creating an auto-associative knowledge base. This can be accomplished by storing a set of statements about the environment in form of bi-polar vectors and information can be retrieved at any time. The biggest benefit of the system is definitely the associative nature of storing information; therefore, to acquire knowledge from the base, an incomplete partial clue is enough as an input. From the network's perspective, it does not matter what patterns are stored, the system can potentially store any kind of information. The only limitation is the memory complexity.

In fact, Hopfield network does not seem to be an optimal memory system, at least if it comes to storage space. Each neuron has to be connected with any other, and this generates plenty of data for which space must be allocated. If connections are represented by a matrix w_{ij} where $w_{ij} = w_{ji}$ and $w_{ii} = 0$, then for n neurons all synaptic connection can be presented in a matrix $n \times n$. Thus, the memory complexity is $O(n^2)$. Another fact is that usually for a pattern with a size m, there is need to create a network with exactly m neurons. Storage space for synaptic connections grows too fast and can exceed physical limitations. To imagine this situation, pattern of size 100 can be taken, e.g. a square picture size of 10 x 10. This means that it is required to create a network with 100 neurons and this produces an enormously huge amount of connections. The easiest way is to create a square matrix with dimensions 100 x 100. This is 10 000 units, each unit should be presented in at least 4 bytes as float number. Thus 40 000 bytes must be allocated, which is around 39 KB. What if a bigger pattern is taken, e.g. a picture of size 100 x 100 pixels, according to the current model, requires about 380 MB.

What about colours? Colours are stored in the file as additional information for each pixel. For example, the bitmap needs 24 bits, three bytes to be able to represent a satisfying number of colours. This only multiplies the amount of neurons in a network.

Despite the fact that the network requires much space, there are ways to optimize it. However, there are no optimizations implemented in current model. One idea is to cut a bigger pattern, e.g. 1000x1000 for many smaller parts such as 100x100; this will generate 100 patterns from one bigger pattern. The next step is to learn the network with this set of data. The problem might be with the accuracy of memory retrieval for that amount of data.

REFERENCES

Achille C. Varzi. 1998. *Basic Problems of Mereotopology*. Accessed on 15 April 2018. Retrieved from:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.6.9025&rep=rep1&type=pdf

Widrow B. 1960. An Adaptive Adaline Neuron using chemical Memistores. Accessed on 23 April 2018. Retrieved from:
http://www-isl.stanford.edu/~widrow/papers/t1960anadaptive.pdf

Crevier D. 1993. *AI: The Tumultuous Search for Artificial Intelligence*. New York: BasicBooks

Baader F., McGuinness D., Nardi D. & Patel-Schneider P. *The Description Logic Handbook – Theory, Implementation and Applications.*

BAM. Bi-Directional Associative Memory. Accessed on 28 April 2018. Retrieved from:
https://uomustansiriyah.edu.iq/media/lectures/5/5_2017_02_28!06_30_52_PM.pdf

Belanche L.A. 2011. Some applications of MLPs trained with backpropagation. Accessed on 25 April 2018. Retrieved from:
http://www.cs.upc.edu/~belanche/Docencia/apren/2009-10/Excursiones/Some%20Applications%20of%20Bprop.pdf

BSB Model. Physics. Brain-State-in-Box (BSB) Model. Accessed on 28 April 2018. Retrieved from:
https://www.physics.nus.edu.sg/~phywjs/CZ3205/Notes9_2.htm

Calosi C. 2016. *Composition, Identity, and Emergence.* Vol.25. No.3. Accessed on 16 April 2018. Retrieved from:
http://apcz.umk.pl/czasopisma/index.php/LLP/article/view/LLP.2016.010/8763

Cycorp®. Knowledge Ontology. OpenCyc. Accessed on 28 February 2018. Retrieved from:
http://cyc.com

Gibbs W., Wilson E.B. 1960. Vector Analysis. Yale University Press.

DevCyc. Knowledge Ontology. OpenCyc. Accessed on 28 February 2018. Retrieved from:
http://dev.cyc.com

Enderton H. B. 1977. *Elements of Set Theory*. New York: Academic Press.

Fiesler E. 1996. *Neural Network Topologies*. Accessed on 20 April 2018. Retrieved from:
https://pdfs.semanticscholar.org/8e5c/f14d4cbe33b0f29e290cc87d2bce94e36a1c.pdf

Graupe D. 2007. *Principles of Artificial Neural Networks*. 2nd edition. Singapore: World Scientific Publisihing.

Gurney K. 1997. *An introduction to neural networks*. 1st edition. London: UCL Press.

Hilbert D. 1950. *The Foundations of Geometry*. Reprint edition. Illinois: The Open Court Publishing Co.

Hilbert D., Ackermann W. 1950. *The Principles of Mathematical Logic*. 2nd edition. New York: Chelsea Publishing Company.

Hopfield J. J. 2007. Hopfield networks. Scholarpedia. Accessed on 15 February 2018. Retrieved from:
http://www.scholarpedia.org/article/Hopfield_network

Hrolenok B. 2009. Recurrent Neural Networks. Accessed on 27 April 2018. Retrieved from:
https://www.cc.gatech.edu/~bhroleno/rnn_slides.pdf

Jakus G., Milutinović V., Omerović S. & Tomažič S. 2013. *Concepts, Ontologies, and Knowledge Representation*. 1st edition. New York: Springer.

Janakiram MSV. 2018. *In The Era Of Artificial Intelligence, GPUs Are The New CPUs*. Accessed 29 April 2018. Retrieved from:
https://www.forbes.com/sites/janakirammsv/2017/08/07/in-the-era-of-artificial-intelligence-gpus-are-the-new-cpus/#71d81fab5d16

Keating J. 1993. *Hopfield networks, neural data structures and the nine flies problem: neural network programming projects for undergraduates*. ACM SIGCSE Bulletin. Vol.25. No. 4: New York.

Krose B., Van der Smagt P. 1996. *An Introduction to Neural Networks*. 8th edition. Amsterdam: University of Amsterdam.

Lyapunov Function. Scholarpedia. 2011. Accessed on 15 February 2018. Retrieved from:
http://www.scholarpedia.org/article/Lyapunov_function

Mohri M., Rostamizadeh A. 2012. *Foundations of Machine Learning*. 1st edition. London: The MIT Press.

Oh C., Hui S. & Żak S. H. 2005. *The Generalized Brain-State-in-s-Box (gBSB) Neural Network: Model, Analysis, and Applications.* Accessed on 28 April 2018. Retrieved from:
https://engineering.purdue.edu/~zak/Microsoft%20PowerPoint%20-%20Pelincec_slides.pdf

Peretto P. 1992. *An Introduction to the Modeling of Neural Networks*. 1st edition. New York: Cambridge University Press.

Poole D. L, Mackworth A. K. 2010. *Artificial Intelligence - Foundations of Computational Agents*. 1st edition. New York: Cambridge University Press.

Brown, Markham F. 2003. Boolean Reasoning: The Logic of Boolean Equations. 1st edition. Kluwer Academic Publishers.

RAII Cppreference. 2018. Accessed 3 May 2018. Retrieved from:
http://en.cppreference.com/w/cpp/language/raii

Redmon J., Divvala S., Girshick R. & Farhadi A. 2015. *You Only Look Once: Unified, Real-Time Object Detection.* Accessed on 26 April 2018. Retrieved from: https://pjreddie.com/media/files/papers/yolo.pdf

Reznik A.M., Dziuba D.A. 2009. *Dynamic Associative Memory, Based on Open Recurrent Neural Network*. Proceedings of International Joint Conference on Neural Networks. Atlanta, Georgia, USA.

Russell S., Norvig P. 2010. *Artificial Intelligence A Modern Approach*. 3rd edition. New Jersey: Pearson Education.

Simson P. 2000. *Parts: A Study in Ontology*. 1st edition. Clarendon Press.

Sompolinsky H. 2013. *Introduction: The Perceptron. MIT*. Accessed 23 April 2018. Retrieved from: http://web.mit.edu/course/other/i2course/www/vision_and_learning/perceptron_notes.pdf

Stewart I., Tall D. 2015. *The Foundations of Mathematics*. 2nd edition. United Kingdom: Oxford University Press.

Tchircoff A. 2017. *The mostly complete chart of Neural Networks. Towards Data Science*. Accessed 20 April 2018. Retrieved from: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

UML. 2018. Accessed 1 May 2018. Retrieved form: http://www.uml.org

Van Atteveldt W. 2008. *Semantic Network Analysis – Techniques for Extracting, Representing, and Quering Media Content*. Charleston: BookSurge Publishers.

Van Harmelen F., Lifschitz V., Porter B. 2008. *Foundations of Artificial Intelligence – Handbook of Knowledge Representation*. 1st edition. Amsterdam: Elsevier B. V.

Wang H., Wu Y., Du K.-L. & Zhang B. 2014. *Recurrent Neural Networks: Associative Memory and Optimization*. ISSN: JITSE - an open access journal. Vol. 1, No. 2: Enjoyor Labs, Enjoyor Inc., Hangzhou, China.

Web Ontology Language (OWL). W3C. Accessed on 15 March 2018. Retrieved form: https://www.w3.org/OWL/

Web Ontology Language: OWL. ResearchGate. Accessed on 25 March 2018. Retrieved from: https://www.researchgate.net/publication/2844629_Web_ontology_language_OWL

Yeruva S., Murty P. S. & Prasad B. 2010. *A Study on Associative Neural Memories*. International Journal of Advanced Computer Science and Applications, Vol. 1, No. 6: Institute of Science and Technology, Kakinada, Andhra Pradesh, India.