SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Chemical and Food Technology

Reg. No.: FCHPT-5414-82048

Decentralized Machine Learning and Optimization

Master thesis

2020 **Bc. Kristína Fedorová**

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Chemical and Food Technology

Reg. No.: FCHPT-5414-82048

Decentralized Machine Learning and Optimization

Master thesis

Study programme: Automation and Information Engineering in Chemistry and Food Industry
Study field: Cybernetics
Training workplace: Institute of Information Engineering, Automation and Mathematics
Thesis supervisor: doc. Ing. Michal Kvasnica, PhD.

Bratislava 2020 **Bc. Kristína Fedorová**

Slovak University of Technology in Bratislava
Institute of Information Engineering, Automation and Mathematics

Faculty of Chemical and Food Technology
Academic year: 2019/2020
Reg. No.: FCHPT-5414-82048

STU FCHPT

MASTER THESIS TOPIC

Student:	**Bc. Kristína Fedorová**
Student's ID:	82048
Study programme:	Automation and Information Engineering in Chemistry and Food Industry
Study field:	Cybernetics
Thesis supervisor:	doc. Ing. Michal Kvasnica, PhD.
Workplace:	Department of information egineering and process control (IIEAM FCHPT)

Topic: **Decentralized Machine Learning and Optimization**

Language of thesis: English

Specification of Assignment:

The objective of the thesis is to design and to implement a system that will be able to process machine learning and optimization tasks in a decentralized environment where multiple computational units collaborate to achieve the joint goal. These units will communicate with each other as to iteratively converge to the global optimum of the complex problem. The designed system will be applied to machine learning with guaranteed privacy where the training data never leave the local device.

Assignment procedure from: 17. 02. 2020

Date of thesis submission: 07. 06. 2020

Bc. Kristína Fedorová
Student

doc. Ing. Michal Kvasnica, PhD.
Head of department

prof. Ing. Miroslav Fikar, DrSc.
Study programme supervisor

Honour Declaration

I declare that the submitted diploma thesis was completed on my own, in cooperation with my supervisor, with the help of professional literature and other information sources, which are cited in my thesis in the reference section.

As the author of my diploma thesis, I declare that I didn't break any third party copyrights.

Signature

Acknowledgment

I would like to thank my thesis supervisor doc. Ing. Michal Kvasnica, PhD., for providing guidance and feedback throughout this thesis and for his support and encouragement.

Bc. Kristína Fedorová

Bratislava, 2020

Abstract

The aim of this thesis is to introduce the distributed optimization algorithms and their application in a decentralized way, where the solution of the optimization problem is reached by dividing the calculations to different computation units. Different types of algorithms, such as ADMM or ALADIN are defined, in order to use them in machine learning. The training process is described and explained on an artificial neural network example. The thesis also provides the result of applying several distributed and decentralized optimization algorithms on linear and nonlinear mathematical problems and their comparison in convergence rate and computational burden. Finally, the same approach is used on model evaluation in machine learning.

Keywords: distributed optimization; decentralized optimization; ADMM; ALADIN; machine learning; artificial neural network

Abstrakt

Cieľom práce je predstaviť algoritmy distribuovanej optimalizácie a ich aplikáciu decentralizovaným spôsobom, kde riešenie optimalizačného problému je dosiahnuté rozdelením výpočtov medzi rôzne výpočtové jednotky. V práci sú taktiež definované rôzne typy algoritmov, akými sú napríklad ADMM alebo ALADIN, za účelom ich využitia v strojovom učení. Trénovací proces, ktorý je súčasťou stojového učenia, je popísaný a vysvetlený na umelých neurónových sieťach. Práca obsahuje zhodnotenie aplikovania niekoľkých distribuovaných a decentralizovaných algoritmov na lineárne ale aj nelineárne matematické problémy a ich porovnanie v rýchlosti konvergencie a výpočtovej záťaži. Nakoniec je rovnaký prístup aplikovania a porovnania použitý aj pri procesene trénovania v strojovom učení.

Kľúčové slová: distribuovaná optimalizácia; decentralizovaná optimalizácia; ADMM; ALADIN; strojové učenie; umelé neurónové siete

Contents

Chapter 1

Introduction

In recent years, nonlinear optimization problems arise in various fields. Economics, banking, computer science, automotive industry, energy industry, process control, etc. The most common solution to such problems is to use model predictive control (MPC). Nowadays, a lot of approaches based on MPC were invented, in order to find the global optimum of the nonlinear problem very precisely and as fast as possible. But we are facing large-scale and more complex problems every day, where the application of MPC is insufficient, because of high computation time and low memory space.

Fortunately, these problems are often structured to have separable objective and constraints in the form

$$\begin{aligned} \min_{x_i,\ldots,x_N} \quad & \sum_{i=1}^{N} f_i(x_i) \\ s.t. \quad & \sum_{i=1}^{N} A_i x_i = b, \end{aligned} \tag{1.1}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ can be convex or nonconvex functions and $x_i \in \mathbb{R}^n$, $i = 1, \ldots, N$ are a vectors of optimized variables. Matrices $A_1, \ldots, A_N$, and vector b are given and they represent the model dependencies between subsystems [10]. Based on this structure, the optimization problem can be divided into a large number of interconnected subsystems (agents) in the network. The cooperation between agents is required to achieve a desired global solution.

Since the optimization problem is distributed among agents, it is called the distributed optimization algorithm. This approach is known for decades, but mainly in a centralized way. It means, the agents have connection only with one central unit, which stores the information from agents and provides necessary calculations. But the centralized framework has its limitations such as high computational burden, single point of failure, limited flexibility, and substantial communication requirement. This is the reason, why decentralized optimization is gradually used in large-scale nonlinear problems.

In decentralized algorithms, each agent holds the estimation of the optimized variable and the update of its value is performed based on the information received from its neighbors and its own information. This information exchange process iteratively leads to a global solution. Modern algorithms for distributed and decentralized optimization is often based on dual decomposition or augmented Lagrangians and the method of multipliers.

Dual decomposition (DD) algorithms are based on solving the decoupled optimization problem with separable, strictly convex cost function and separable constraints. The way to solve such a problem is to use the gradient ascent method as suggested in [14] or [16]. An alternative to DD algorithms is the alternating direction method of multipliers (ADMM), originally introduced in [7]. In comparison with DD, the construction of dual function is not required, but the solution is found by using the augmented Lagrangian form.

In addition, this paper deals not only with algorithms for a convex objective but also the algorithms, which have been developed for solving the distributed non-convex optimization problems. One of these types of algorithms is recently proposed augmented Lagrangian based alternating direction inexact Newton method (ALADIN). ALADIN has a lot of benefits in comparison with ADMM and they are discussed in the paper.

The distributed and decentralized optimization is not only used for solving large-scale problems, but its implementation arises also in the field of machine learning. We live

in a data era. Our phones, tablets and laptops are full of data, which can be very useful not only for companies and their marketing strategies but mainly for us, for our finance or our health. The sharing of the data could lead to an improvement in many fields. However, the privacy concerns have appeared, in order to prevent a misusage of the information. That is why the rapid development in decentralized machine learning has been realized.

This represents the motivation of the thesis. To understand the distributed algorithms and be able to use them in a decentralized way for solving the problems starting with simple convex problems and ending with large-scale non-linear and nonconvex problems. But mainly, to be able to apply this knowledge in machine learning in order to keep privacy.

Chapter 2

Distributed Optimization

To understand the definition of decentralized optimization, we will first introduce the concept of distributed optimization. The distributed approach of solving optimization problems is based on the decomposition of a problem (if it is possible) to several other subproblems with certain cost functions. The subproblems are distributed to the agents (calculation units, processors, etc.), where computations are performed synchronously and the partial results are shared on the network, between agents [19]. This result exchange iteratively leads to the global solution of the initial optimization problem.

Centralized optimization is a coordinated policy to solve a problem that considers the system under study as a whole one system. Generally, it solves the optimization problem in the form

$$\begin{aligned} \min_{x} \quad & f(x) \\ s.t. \quad & Ax = b, \end{aligned} \tag{2.1}$$

which is explained further in Sec. 2.1. On the other hand, a no-coordinated policy to solve a problem considering the system under study as a number of sub-systems which are optimized separately is called decentralized optimization. In comparison with (2.1),

a definition of the decentralized optimization problem can be constructed as

$$\begin{aligned} \min_{x} \quad & \sum_{i=1}^{N} f_i(x_i) \\ s.t. \quad & \sum_{i=1}^{N} A_i x_i = b, \end{aligned} \tag{2.2}$$

where constraints are defined in form, where variables are not interdepent [12].

2.1 Dual Ascent

The dual ascent method is the optimization algorithm, which provides some useful background and motivation to algorithms of distributed optimization. We consider the convex optimization problem with equality constraints in the form

$$\begin{aligned} \min_{x} \quad & f(x) \\ s.t. \quad & Ax = b, \end{aligned} \tag{2.3}$$

with variable $x \in \mathbb{R}^n$, where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $f : \mathbb{R}^n \to \mathbb{R}$ is convex.

The Lagrangian for problem (2.3) is

$$L(x, \lambda) = f(x) + \lambda^\top (Ax - b), \tag{2.4}$$

and the dual function is

$$g(\lambda) = \inf_{x} L(x, \lambda) = \inf_{x} \left(f(x) + \lambda^\top (Ax - b) \right), \tag{2.5}$$

where λ is the dual variable or Lagrange multiplier. The dual problem of (2.3) is

$$\max_{\lambda} g(\lambda), \tag{2.6}$$

with variable $\lambda \in \mathbb{R}^m$. The optimal values of primal and dual problems are the same, if we assume that strong duality holds. We can recover a primal optimal variable $x^\star$ from a dual optimal variable $\lambda^\star$ as

$$x^\star = \arg\min_{x} L(x, \lambda^\star). \tag{2.7}$$

Dual optimal point is obtained as a solution of (2.6)

$$\lambda^\star = \arg\max_\lambda L(x^\star, \lambda). \tag{2.8}$$

This solution cannot be used, because it could lead to infinity. So in the dual ascent method, the dual problem is solved by using gradient ascent. If $g(\lambda)$ is differentiable, then the gradient $\nabla g(\lambda)$ can be evaluated as $\nabla g(\lambda) = Ax^+ - b$, where $x^+ = \underset{x}{\operatorname{argmin}}\ L(x, \lambda)$. It is obvious that, $\nabla g(\lambda)$ is then the residual for equality constraint. The dual ascent method consists of the following updates

$$x^{k+1} = \arg\min_x L(x, \lambda^k), \tag{2.9a}$$

$$\lambda^{k+1} = \lambda^k + \alpha^k(Ax^{k+1} - b), \tag{2.9b}$$

where the superscript k is the iteration counter and $\alpha^k > 0$ is a step size. The dual variable λ can be interpreted as a vector of prices. As we can see, with appropriate choice of α^k, the dual function increases in each step, i.e., $g(\lambda^{k+1}) > g(\lambda^k)$ [3], so this algorithm is called dual ascent. The biggest benefit of the dual ascent method is that it can lead to distributed algorithms.

2.2 Dual Decomposition

Dual decomposition relies on the observation that many optimization problems can be decomposed into two or more subproblems [16]. Let suppose, that the objective f is separable, meaning that

$$f(x) = \sum_{i=1}^{N} f_i(x_i), \tag{2.10}$$

where $x = (x_1, \ldots, x_N)$ and the variables $x_i \in \mathbb{R}^{n_i}$ are subvectors of vector x. Also the matrix A can be separated into $A = [A_1, \ldots, A_N]$, so $Ax = \sum_{i=1}^N A_i x_i$ and the Lagrangian can be written as

$$L(x, \lambda) = \sum_{i=1}^{N} L_i(x_i, \lambda) = \sum_{i=1}^{N} \left(f_i(x_i) + \lambda^\top A_i x_i\right) - \frac{1}{N}\lambda^\top b, \tag{2.11}$$

which is also separable in x. This can lead to the optimization problem, that can be splitted into N suproblems. Then these subproblems are parallelly solved. Explicitly, the algorithm is

$$x_i^{k+1} = \arg\min_{x_i} L_i(x_i, \lambda^k), \tag{2.12a}$$

$$\lambda^{k+1} = \lambda^k + \alpha^k(Ax^{k+1} - b). \tag{2.12b}$$

The x-minimization step (2.12a) is realized independently, in parallel, for each iteration k and each agent i. Once the dual variable λ^{k+1} is computed in dual update step (2.12b), it is immediately distributed back to agents, which again individualy perform step (2.12a). In dual decomposition, the dual ascent method is also used on decomposed optimization problem [7].

2.3 Augmented Lagrangians and the Method of Multipliers

Augmented Lagrangian methods were developed in part to improve the dual ascent method. For now, the distributed algorithms assumed that f is strictly convex. The augmented Lagrangian methods should bring robustness to distributed algorithms without mentioned assumptions. The augmented Lagrangian for (2.3) has a form

$$L_\rho(x, \lambda) = f(x) + \lambda^\top(Ax - b) + \frac{\rho}{2}||Ax - b||_2^2, \tag{2.13}$$

where $\rho > 0$ stands for the penalty parameter. If $\rho = 0$, then L_0 is the standard Lagrangian for the problem. We can afford to add the term $||Ax - b||_2^2$ in (2.13), because the augmented Lagrangian is equal to original for any feasible x. The benefit of including the penalty parameter is that dual function

$$g_\rho = \inf_x L_\rho(x, \lambda), \tag{2.14}$$

can be differentiable under mild conditions in comparison to the original problem.

Applying dual ascent to the introduced, modified problem construct the algorithm

$$x^{k+1} = \arg\min_x L_\rho(x, \lambda^k), \tag{2.15a}$$

$$\lambda^{k+1} = \lambda^k + \rho(Ax^{k+1} - b), \tag{2.15b}$$

which is known as the method of multipliers. It can be seen, that the algorithm is almost the same as the original dual ascent. But instead of standard Lagrangian L_0, the augmented Lagrangian L_ρ is used and the step size α^k is substituted by penalty parameter ρ.

The benefit of this method is in good convergence properties, but it also has its limits. Even the f is separable, the augmented Lagrangian is not and that is the reason, why the decomposability is lost in the method of multipliers. We will see how to address this issue next.

2.4 Alternating Direction Method of Multipliers

Alternating Direction Method of Multipliers (shortly ADMM) is an algorithm that combines the good convergence properties of the method of multipliers with the decomposability of dual decomposition. The algorithm solves problems in the form

$$\begin{aligned} \min_x \quad & \sum_{i=1}^{N} f_i(x_i) \\ s.t. \quad & Ax = b, \end{aligned} \tag{2.16}$$

that can be modified as follows

$$\begin{aligned} \min_x \quad & \sum_{i=1}^{N} f_i(x_i) \\ s.t. \quad & Ax + By = b, \end{aligned} \tag{2.17}$$

where $y = (y_1, \ldots, y_N)$ is vector of initial guesses which are updated in each iteration. In general, the y represents the global optimal solution for the current iteration. As in

the method of multipliers, we form the augmented Lagrangian

$$L_\rho(x, y, \lambda) = f(x) + \lambda^\top (Ax + By - b) + \frac{\rho}{2}||Ax + By - b||_2^2. \tag{2.18}$$

ADMM consists of the iterations (for specific details see Algorithm 1)

$$x^{k+1} = \arg\min_x L_\rho(x, y^k, \lambda^k), \tag{2.19a}$$

$$y^{k+1} = \arg\min_y L_\rho(x^{k+1}, y, \lambda^k), \tag{2.19b}$$

$$\lambda^{k+1} = \lambda^k + \rho(Ax + By - b), \tag{2.19c}$$

where $\rho > 0$. The algorithm is very similar to the method of multipliers, but it consists of one additional step, y-minimization step. In some application, this step can be replaced by evaluating the average value of all local optimized variables. Then, the part $||Ax + By - b||_2^2$ can be also separable, what indicates the improvement in comparison to method of multipliers.

Algorithm 1: ADMM

Input: Initial guesses $y \in \mathbb{R}^n$ and $\lambda_i \in \mathbb{R}^m$ and a numerical tolerance $\varepsilon > 0$.

Repeat:

1. Choose a penalty parameter $\rho > 0$ and solve for all $i \in (1, \ldots, N)$ the decoupled nonlinear optimization problems (NLPs)
$$\min_{x_i} \left(f(x_i) + \lambda_i^\top (A_i x_i + B_i y - b) + \frac{\rho}{2}||A_i x_i + B_i y - b||_2^2 \right). \tag{2.20}$$
2. If $||\sum_{i=1}^N (A_i x_i + B_i y - b)||_1 \leq \varepsilon$, terminate with $x^\star = y$ as a numerical solution.
3. Solve the coupled quadratic problem
$$\min_{y^+} \left(\lambda_i^\top (A_i x_i + B_i y^+ - b) + \frac{\rho}{2}||A_i x_i + B_i y^+ - b||_2^2 \right). \tag{2.21}$$
4. Implement the dual gradient steps
$$\lambda_i^+ = \lambda_i + \rho(A_i x_i + B_i y^+ - b). \tag{2.22}$$
5. Update the iterates $y \leftarrow y^+$ and $\lambda \leftarrow \lambda^+$ and continue with Step 1.

In ADMM x and y are updated in an alternating fashion, which accounts for the term alternating direction [3]. The separation of algorithm to three steps is what allows for decomposition. The limitations of ADMM are that it may be divergent, if f_i are nonconvex functions and also the convergence rate of this method is very scaling dependent.

The ADMM algorithm was applied on various types of problems, which needed to be divided in subproblems and the results are described in experimental part of thesis.

2.5 ALADIN

Augmented Lagrangian based Alternating Direction Inexact Newton Method (shortly ALADIN) concerns structured convex optimization problems of the form

$$\begin{aligned} \min_{x} \quad & \sum_{i=1}^{N} f_i(x_i) \\ s.t. \quad & \sum_{i=1}^{N} A_i x_i = b. \end{aligned} \tag{2.23}$$

It also can be transformed in form (2.17). We define the Augmented Lagrangian for each iteration i as in ADMM

$$L_\rho(x, y, \lambda) = f(x) + \lambda^\top (Ax + By - b) + \frac{\rho}{2} ||Ax + By - b||_\Sigma^2, \tag{2.24}$$

where $x = (x_1, \ldots, x_N)$ represents local optimization variable, $y = (y_1, \ldots, y_N)$ is vector of initial guesses which are upgraded in each iteration according Algorithm 2. And

$$||Ax + By - b||_\Sigma^2 = (Ax + By - b)^\top \Sigma (Ax + By - b), \tag{2.25}$$

where Σ is positive semi-definite matrix [10].

It is obvious that the cost of one ALADIN iteration is exactly the same as the cost of one ADMM iteration since the only difference between these two algorithms is that ALADIN maintains only one dual variable $\lambda \in \mathbb{R}^m$ that is updated in a slightly different manner than the dual variables $\lambda_1, \ldots, \lambda_N \in \mathbb{R}^m$ from Algorithm 1.

Algorithm 2: ALADIN

Input: Initial guesses $y \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}^m$ and a numerical tolerance $\varepsilon > 0$.

Repeat:

1. Choose a penalty parameter $\rho > 0$ and positive semi-definite matrices Σ_i and solve for all $i \in (1, \ldots, N)$ the decoupled NLPs
$$\min_{x_i} \left(f(x_i) + \lambda_i^\top (A_i x_i + B_i y - b) + \frac{\rho}{2} ||A_i x_i + B_i y - b||^2_{\Sigma_i} \right). \tag{2.26}$$
2. If $|| \sum_{i=1}^N (A_i x_i + B_i y - b) ||_1 \leq \varepsilon$ and $\rho ||\Sigma_i (A_i x_i + B_i y - b)||_1 \leq \varepsilon$, terminate with $x^\star = y$ as a numerical solution.
3. Compute $g_i = \nabla f_i(x_i)$, choose $H_i \approx \nabla^2 f_i(x_i)$, and solve coupled quadratic problem
$$\begin{aligned} \min_{\Delta x} \quad & \sum_{i=1}^N \left(\frac{1}{2} \Delta x_i^\top H_i \Delta x_i + g_i^\top \Delta x_i \right) \\ s.t. \quad & \sum_{i=1}^N A_i (x_i + \Delta x_i) = b \quad | \quad \lambda^+. \end{aligned} \tag{2.27}$$
4. Update the iterates $y \leftarrow x + \Delta x$ and $\lambda \leftarrow \lambda^+$ and continue with Step 1.

For many cases, the choice of g and H is introduced in Algorithm 2. But the selection must be different, if ALADIN is applied to the problem with private and sensitive data. Therefore, the gradient and hessian matrix for our test cases were chosen randomly and tuned to achieve a satisfactory result.

The advantages of ALADIN in comparison with ADMM were introduced in [10]. Shortly, ALADIN provides a significant improvement in convergence properties, even with the non-convex optimization problems.

2.6 Decentralized Optimization

First we start with centralized optimization. In a centralized way of distributed optimization, the local optimum of each agent is shared among the network and send to the central unit (Fig. 2.1A.). The central unit provides global optimum (based on global optimization, averaging etc.) and its distribution back to the agents.

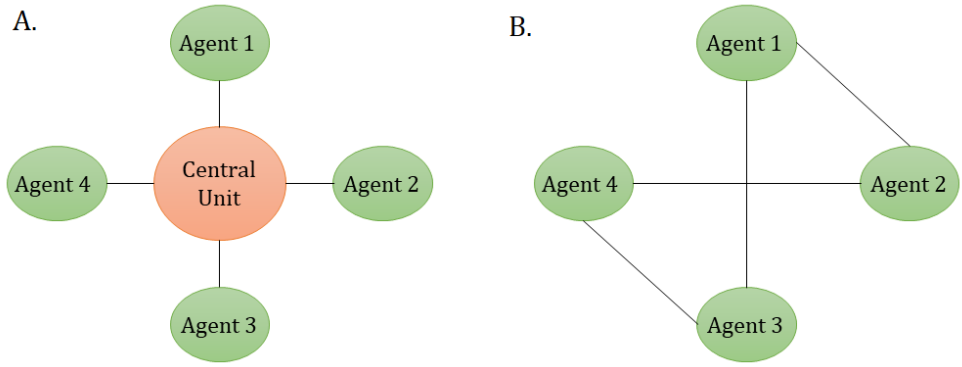

Figure 2.1: A. Centralized configuration, B. Decentralized configuration

On the other hand, in a decentralized way, there is no central unit. The agents are connected in some pre-defined pattern (for instance Fig. 2.1B.) and share the local result only with their connected neighbors. The global result is then calculated in each agent, based on the information and local results from mentioned neighbors [18]. To understand the motivation of choice the decentralized approach over centralized, we will discuss the pros and cons.

The centralized optimization is an algorithm with fast convergence (in comparison with decentralized). The reason is, the central unit has knowledge about local optima from all agents and provides the global result based on this information. More information, better convergence. But with more information the memory demand and computation burden increase. Because of these disadvantages of centralized optimization, we decided

to focus on decentralized optimization.

Table 2.1: Pros and cons of centralized optimization

Centralized optimalization	
Advantages	Disadvantages
+ fast convergence	- memory footprint
	- computation burden
	- one point of failure
	- price

By selecting the decentralized approach, we decided to give up the fast convergence properties. The information exchange works only locally and that is the explanation of slower convergence. Hand in hand with less information, the computation burden and memory demand decrease [20].

Table 2.2: Pros and cons of decentralized optimization

Decentralized optimalization	
Advantages	Disadvantages
+ memory footprint	- slower convergence
+ computation burden	- price

The price question is contentious. Each iteration cost something, so the faster convergence, the less price. But it also depends on the number of connections between agents or central units and agents. The difference between the two mentioned approaches will be also discussed in the experimental part of the thesis.

Chapter 3

Machine Learning

Machine learning is an application of artificial intelligence that make systems to be able to automatically learn and improve from experience without being explicitly programmed. The main goal is to find a solution to the problem without any human intervention. The learning process is based on training data, which has to be provided. A lot of learning approaches and models exist. In this chapter, we will introduce the deep learning approach to the artificial neural network model.

3.1 Types of Machine Learning

In recent years, rapid development in machine learning has been observed. The behavior of machine learning has been tested on various tasks and problems based on different learning approaches. With increasing discoveries and improvements in the field of machine learning, the numbers of approaches, model structures, and even the types of performed tasks have arisen and the existing types of machine learning have been divided into several groups as shown in Tab. 3.1.

Table 3.1: Categories of machine learning

MACHINE LEARNING		
1. Performed task	2. Learning approach	3. Model structure
classification regression	supervised unsupervised reinforcement	neural networks decision trees support vector machines

3.1.1 Machine Learning Tasks

Even though there are many tasks, in various types of problems, that can be handled by machine learning, the tasks are divided only into two main groups.

- Classification tasks

The classification predictive modeling task is based on approximating a mapping function from input variables to discrete output variables. In general, the mapping function predicts the class or category for a given observation. It is common for classification models to predict continuous values, which can be reffered as the probability of a given example belonging to each output class. For instance, an email can be classified as *spam* or *not spam*. The machine learning provides the prediction model, which can distinguish between these two email classes. The output carries information about probability, to which class a received mail belongs. To verify the skill of model the classification accuracy has been introduced. It is a percentage of correctly classified examples out of all predictions made.

- Regression tasks

On the other hand, the regression predictive modeling task is based on approximating a mapping function from input variables to continuous output variables. The output

variable can be any real number, such as integer or floating-point value. These often represent the quantities or other functional dependences [6]. A simple example is the estimation of salary depending on the age of an employee. There are no categories, but the dependence between age and salary is found, based on real data. This predictive model can be very beneficial in healthcare or process control. The skill of the model is reported as an error in the provided predictions. An error can be evaluated as an absolute value between expected and predicted output or the square of this difference, etc.

3.1.2 Machine Learning Approaches

Nowadays, there are three types of learning approaches for machine learning.

- Supervised learning

Supervised learning is the process of training based on a training input-output pairs set. The main idea is to approximate the mapping function, to provide the desired output for a given example, based on iterative learning and evaluation of error (correctness) at the end of each iteration. Once the model is trained, it can be used to assign output value to any input value even outside the training set. It is called supervised because the process of learning from the training set can also be understood as a teacher supervising the learning process. This approach is widely used in many fields, for example for risk evaluation, fraud detection, or prediction of financial results.

- Unsupervised learning

Another approach is called unsupervised learning. This type of machine learning searches patterns in supplied data set with no pre-defined outputs, in contrast to supervised learning, where outputs are provided. This modeling process was introduced mainly for classification tasks. Literally, the demand for machine learning is to learn,

how to find or evaluate the output. The algorithm consists of logical and mathematical operations, that allow machine learning to identify the differences between inputs and then categorize them. The application of unsupervised learning is mainly in item categorization, clustering customers, and similar item recommendations.

- Reinforcement learning

Reinforcement learning is a subfield of machine learning that teaches an agent, which action to take in a particular environment, in order to maximize rewards over time. In this notation, the agent represents the program we want to train to do a job we specify. The environment is a real or virtual world, in which the agent performs actions. Each action causes a status change in the environment. And the reward is the evaluation of an action, that can be positive or negative and which represents the correctness of the action. The goal is to maximize reward, which can be defined as some cost function. The reinforcement approach is used in a very attractive area of robotics, games, or self-driving cars [2].

3.1.3 Model Structure

In machine learning, we can also decide, which type of structure of the predictive model, we want to use. There are artificial neural networks, where connections are created based on neural networks in the human brain and the aim is to find mapping function, representing the relationship between inputs and outputs. Then decision trees, which classify instances by sorting them based on model, where braches representing the observation about an item and the leaves representing the conclusions about the item's target value. Or supper vector machine that separates two data classes based on the solution of optimization problem [13].

For the purpose is this thesis, we decided to choose an artificial neural network model. Although the majority of machine learning models are suitable for the application

of a decentralized training approach, artificial neural networks are often and widely used these days in a lot of machine learning problems. This is why we will only deal in-depth with neural networks.

3.2 Artificial Neural Network

Artificial neural networks are systems inspired by biological neural networks. Such systems are designed to learn to perform any computation or task. The programming of rules or decision conditions is not necessary. The most important ingredient for neural networks is datasets. The neural network can be trained to provide an expected output based on given input in various industries (medicine, banking, computer science, process control, etc.) [11]. There is only one condition. A sufficient amount of data.

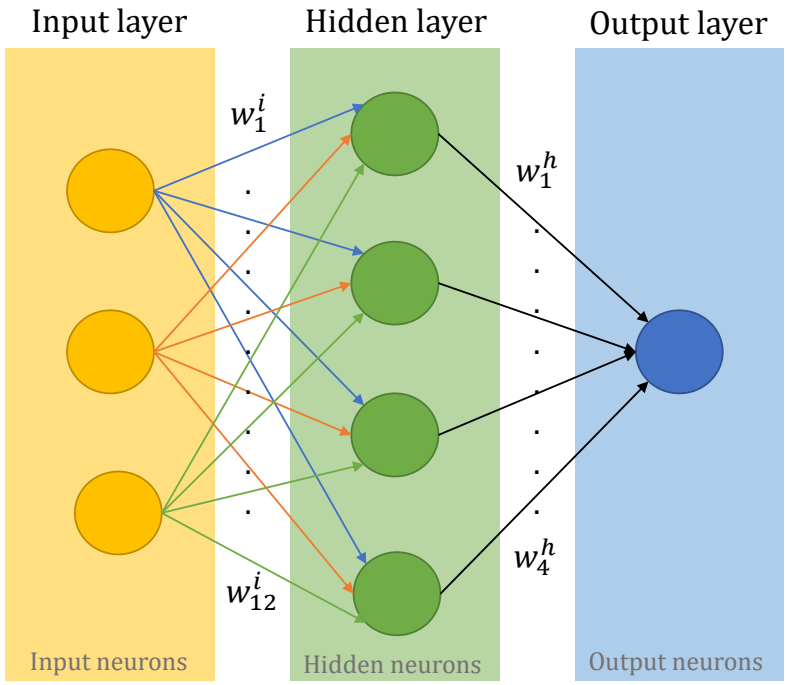

Figure 3.1: The basic structure of neural network

3.2.1 Structure and Calculations

In general, the neural network is composed of computational units called neurons or nodes. They are organized in layers and each neuron has a connection with neurons from different layers (see Fig. 3.1). The neurons in layers with their connections create the network.

To explain how neural networks work, we have to start will layers. There are three types of layers: input, hidden, and output layer. The number of neurons in the input or output layer depends on the size of inputs or outputs from the training dataset respectively. A system with three inputs and one output would have the three neurons in the input layer and one neuron in the output layer. The hidden layer is a special case. The neural network does not have to consist only of one hidden layer. The number of hidden layers and the number of their neurons are not restricted. The appropriate setup is described in [4].

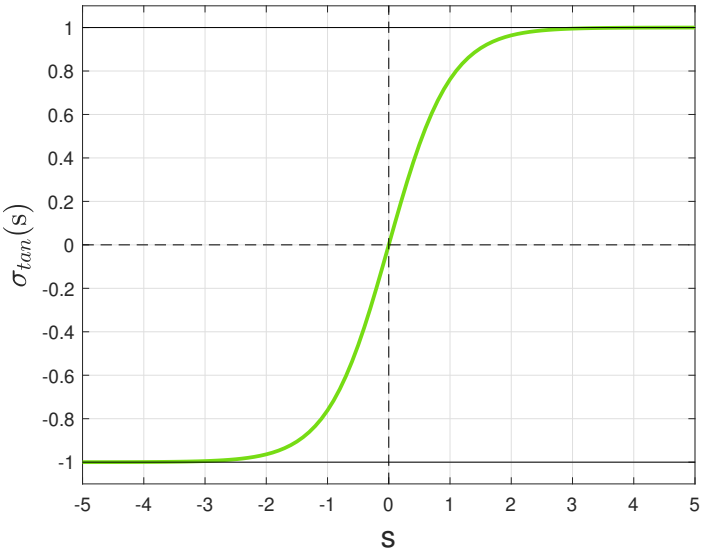

Figure 3.2: The graph of hyperbolic tangent sigmoid function

To deeply understand, how a neural network works, we have to precisely define the variables in it. So, let the n_k^l be the k-th neuron (node) in layer l. Then, let z_k^l be the input vector to n_k^l, computed as

$$z_k^l = (a^{l-1})^\top w_k^l, \tag{3.1}$$

where w_k^l is the vector of weights assigned to node n_k^l, expressing the importance of corresponding inputs to the outputs, and a^{l-1} is the vector of outputs from the previous layer. Now, the term bias will be introduced. Bias b_k^l is the numerical value, unique for each neuron, which is added to the z_k^l. The expression $z_k^l + b_k^l$ represents the argument of the activation function, which provides the output from layer [9].

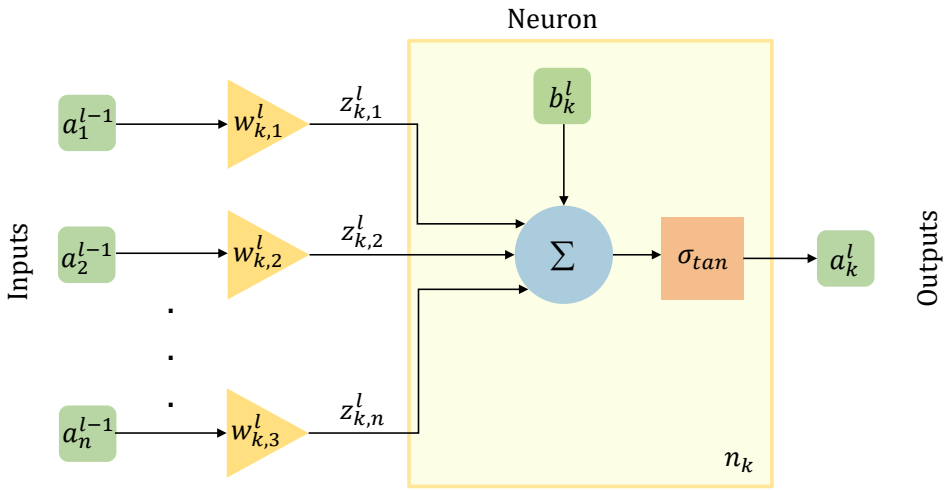

Figure 3.3: Structure of computations in neuron

In general, there are four types of activation (transfer functions). Unit step (threshold), sigmoid, piecewise linear, and Gaussian. The purpose of the transfer function is to translate the input signals to output signals and to keep the output values within specified bounds. We have decided to use the hyperbolic tangent sigmoid transfer

function $\sigma_{tan}(s)$, defined as follows

$$\sigma_{tan}(s) = \frac{2}{1 + \mathrm{e}^{-2s}} - 1. \tag{3.2}$$

According to the (3.2), the result of each neuron in our case is between -1 and 1 (see Fig, 3.2). After applying the activation function, the outcome of node n_k^l is computed as

$$a_k^l = \sigma_{tan}(z_k^l + b_k^l) = \frac{2}{1 + \mathrm{e}^{-2(z_k^l + b_k^l)}} - 1. \tag{3.3}$$

If we consider the input layer as $l = 0$, then a^0 is initialized as $a^0 = x_i$, where x_i is the input vector from given training dataset. Then (x_i, y_i) represents the input-output pair, where $i = 1, \ldots, N$, while N represents the length of training dataset. The calculation are also illustrated in Fig. 3.3.

3.2.2 Training Process

We have just explained the function of each neuron in our network. But how is it possible, the neural network provides the expected output? We will answer the question in this section. But first, we have to understand the training process of the neural net is composed of two stages:

- Forwardpropagation (model evaluation)
- Backpropagation (model training)

3.2.2.1 Forwardpropagation

Forward propagation is the stage, where weights and biases are fixed and inputs can vary, which provides the predicted output. In the model evaluation part, the difference between real outcome from neural network and target outcome is not important. This stage is called feedforward because the output from one layer is used as input of the

next layer, there are no loops involved and the information is only passed forward and never back [15].

The first step in the learning process is to initialize the weights and biases mentioned in the neural network structure. Normally, their values are initialized randomly. Then, the neural net is built and starts with a feedforward step to compute the output. In short, the input layer provides the training set. The input to the (first) hidden layer is obtained as a multiplication of the delivered training set and its weights (similar to (3.1)). The output of this layer is computed as the result of the activation (transfer) function described in (3.2). The result multiplied with another weights set represents the input to the next layer. And the procedure is repeated again until the output is achieved. This is the feedforward step.

The very important operation is to iterate over the whole training set to let the network learn and to evaluate all predicted outputs. Obviously, the outcome of the feedforward step does not match the expected output. So the learning procedure is switched to the second phase called backpropagation.

3.2.2.2 Backpropagation

In the second stage, the model training stage, the inputs, and the predicted outcomes are fixed, while weights and biases vary and the network ends with error evaluation. The sequence of backward steps is executed in parallel with updating the weights and biases based on the values of two parameters. The first is the value of cost function describing the inaccuracy (error) of the result produced by the forward step and the second is the learning rate. The most commonly used cost function is quadratic called mean squared error (MSE)

$$E(X,\theta) = \frac{2}{N}\sum_{i=1}^{N} \|\hat{y_i}(x,\theta) - y_i\|_2^2, \tag{3.4}$$

where $\hat{y}$ is predicted output, weights and biases are collectively denoted as θ and X stands for input-output vector pairs (x_i, y_i). Naturally, the cost function looks simple. But we have to remember, y is computed as a long sequence of the mathematical operations with a huge amount of variables. The aim of the neural network learning is to equal target and computed output. It means, we have to find optimal values of weights and biases by minimizing the cost function as

$$\min_{\theta} E(X, \theta) = \min_{\theta} \frac{2}{N} \sum_{i=1}^{N} \|\hat{y_i}(x, \theta) - y_i\|_2^2. \tag{3.5}$$

Sometimes this optimization problem can be easily solved by calculating the derivates of (3.4). Mostly, the problem includes a lot of optimized variables, which make this computation quite impossible to make. That is the reason, why the gradient descent method is used in neural network training.

In literature, the term backpropagation is referred to as an algorithm for computing gradient. It does not include the information about, how this gradient is used [8]. This is exactly what the gradient descent method serves for. But for the purpose of this paper, we label the whole model training stage as backpropagation.

Gradient descent method is an iterative approach of finding the local minimum of the problem. It is based on the evaluation of the function gradient (or its approximation). The steps, proportional to the negative gradient, are taken (more information in Algorithm 3), in order to reach the optimum.

In the neural network training process the step α, introduced in (3.7), is called learning rate. α defines how fast, we move towards the global optimum. The choice of this parameter is very important. If α is very small, a lot of iterations are needed for reaching the minimum. On the other hand, the big numerical value of the parameter could lead to failure in finding a minimum by passing it.

Algorithm 3: Gradient descent

Input: Initial guesses x_0 and step α and a numerical tolerance $\varepsilon = 10^{-6}$.

Repeat:

1. Evaluate the value of the gradient of function $f(x)$ or its approximation
$$\nabla = \frac{df(x)}{dx}. \tag{3.6}$$
2. If $\nabla \leq \varepsilon$, terminate with $x^\star = x_0$ as a numerical solution.
3. Calculate the x-variable
$$x = x_0 + \alpha\nabla. \tag{3.7}$$
4. Update the iterates $x \to x_0$ and continue with Step 1.

As we mentioned, it is not always possible to evaluate the gradient of $E(X, \theta)$. So, in order to use a gradient descent method, we have to find an efficient way to compute $\nabla E(X, \theta)$. For this purpose, we will introduce the backpropagation algorithm. To simplify the notation, we will consider the bias of neuron n_k as a part of the weights vector

$$W_k^l = (b_k^l, w_{k,1}^l, \ldots, w_{k,n}^l)^\top, \tag{3.8}$$

and also a^l is augmented as

$$A^l = (1, a_1^l, \ldots, a_n^l)^\top. \tag{3.9}$$

That way, multiplication $(A^{l-1})^\top W_k^l$ represents the same expression as $(a^{l-1})^\top w_k^l + b_k^l$.

The backpropagation algorithm is characterized by evaluating subgradients, step by step, in a backwards direction (from the output layer to the input layer). The gradient of the cost function can be evaluated based on chain rule as

$$\nabla E(X, \theta) = \left(\frac{\partial E(X, \theta)}{\partial A^L} \frac{\partial A^L}{\partial Z^L} \frac{\partial Z^L}{\partial A^{L-1}} \frac{\partial A^{L-1}}{\partial Z^{L-1}} \frac{\partial Z^{L-1}}{\partial A^{L-2}} \cdots \frac{\partial A^1}{\partial Z^1} \frac{\partial Z^1}{\partial x_i} \right)^\top, \tag{3.10}$$

where L is the number of the last layer. Some parts can be substituted. The derivation of input Z^L in terms of previous output A^{L-1} is computed as a weight W^L between them. We also simplify the expression $\frac{\partial A^L}{\partial Z^L}$ to derivative of activation function $(\sigma^L)'$.

$$\nabla E(X, \theta) = (W^1)^\top (\sigma^1)' \ldots (W^{L-1})^\top (\sigma^{L-1})' (W^L)^\top (\sigma^L)' \nabla_{A^L} E(X, \theta) \tag{3.11}$$

Then the term layer error is defined as

$$\delta^l = (\sigma^l)'(W^{l+1})^\top \dots (W^{L-1})^\top (\sigma^{L-1})'(W^L)^\top (\sigma^L)' \nabla_{A^L} E(X, \theta), \tag{3.12}$$

and we can evaluate the subgradient for layer l

$$\nabla E^l = \delta^l (A^{l-1})^\top. \tag{3.13}$$

The δ^l can easily be computed recursively as

$$\delta^{l-1} = (\sigma^{l-1})'(W^l)^\top \delta^l. \tag{3.14}$$

Now, we finally have all ingredients to calculate the new weights W^L for each layer, based on gradient descent algorithm (Algorithm 3), starting with the last layer and progressively approach the input layer [17]

$$W^l_{update} = W^l + \alpha \nabla E^l. \tag{3.15}$$

And this is the whole training process. Naturally, the mentioned steps are repeated, until all weights and biases reach their optimal values and the neural network provides the minimal MSE for each training set.

3.3 Decentralized Learning

Since we already know how the artificial neural network training works, we are able to divide this process into several computational units (agents) as shown in Fig. 3.4. That way, the computational burden and memory footprint of one calculation unit are decreased by delegating the part of the calculations to the agents, and the whole procedure can be more effective. Nowadays, this approach is widely studied, because there are even more advantages relating to privacy.

We live in a data era. There are millions, billions, or even more datasets, which can be very useful for machine learning. The data, that are publicly available and some

companies have the right to use them in their machine learning algorithms, come from user searches on the internet, from social media contribution, etc. Still, it is only a small fraction of all data. The rest is the locked private data, that user's (or generally the agents) do not want to share and which stay untapped in users devices, such as mobile phones, tablets, and laptops.

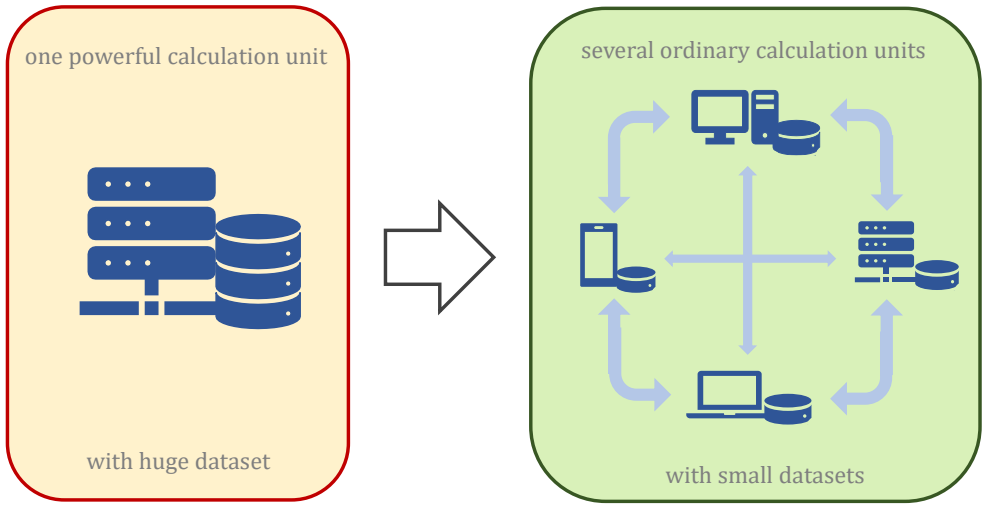

Figure 3.4: Scheme of distribution the calculations from one to several calculation units

No one wants his privacy violated, so in order to keep it but be able to use the sensitive data in the training process, the distributed and decentralized machine learning was introduced. The main idea is simple. Since there is no way to obtain the user private data for the training process, the opposite approach was devised. The training model is provided to the agents, which allows them to run their own learning procedure on their data and to share only partial results (parameters of the model), while the train data never leave the local devices [1].

It works as follows. Let's have a network with K agents. The training dataset X, consisting of input-output pairs (x_i, y_i), where $x_i \in \mathbb{R}^n$ is input vector, $y_i \in \mathbb{R}^m$ is output vector of i-th sample, is created from the K data subsets. Each agent receives the model of the neural network (number of hidden layers and neurons) and is encouraged to start its own learning. After finding the optimal values of biases b and weights w of models by each agent, these parameters are shared among network, the average is computed (the ADMM or ALADIN approach can be used) and the training process starts again with biases and weights initialized as mentioned average. These steps are performed repeatedly until all agent models converge to the same values of b and w.

The shared partial result can be processed in one place, parent agent, or central unit, but it has some disadvantages. If the central unit fails or stops working, the whole training process is interrupted. Therefore, a decentralized approach is increasingly used [5]. This way agents communicate and exchange the information between them without any intermediary or third party. It is called peer-to-peer communication. If any agent stop working, the learning process is not compromised. The convergence rate of both approaches is discussed in Chap. 5, with all graphic and numerical results.

Chapter 4

Application of Distributed Optimization

In this part, we focus on the application of ADMM and ALADIN methods, in order to compare their convergence properties, not only on problems with a smooth convex objective but also with non-smooth and non-convex cost functions. The chapter also contains a discussion dealing with a comparison of a centralized and decentralized approach.

4.1 Data Fit with Affine Function

The problem of fitting the data is a very common and for the needs of the paper, we assume this problem also for the demonstration of the efficiency of distributed optimization. The general form of data fit problem, if we consider the approximation of data by affine function, is defined as follows

$$\min_{a,b} \sum_{i=1}^{N} \left((a_2 x_i + a_1) - y_i\right)^2, \tag{4.1}$$

where a_1 and a_2 are optimized variables and also a_2 represents the slope of the function and a_1 its section. $x = (x_1, \ldots, x_N)^\top$ and $y = (y_1, \ldots, y_N)^\top$ are data sets (see Fig. 4.1) that are asssumed to be given.

In this particular problem, we have to deal with the issue that every agent in the

networked system has its own private data of position and this data are not allowed to share. In this point, the problem (4.1) becomes more complex. The main task is to solve the problem to global optimum on the condition of privacy. For this purpose, the optimization problem has to be divided into a local and global optimization problem, which is the base of decentralized optimization.

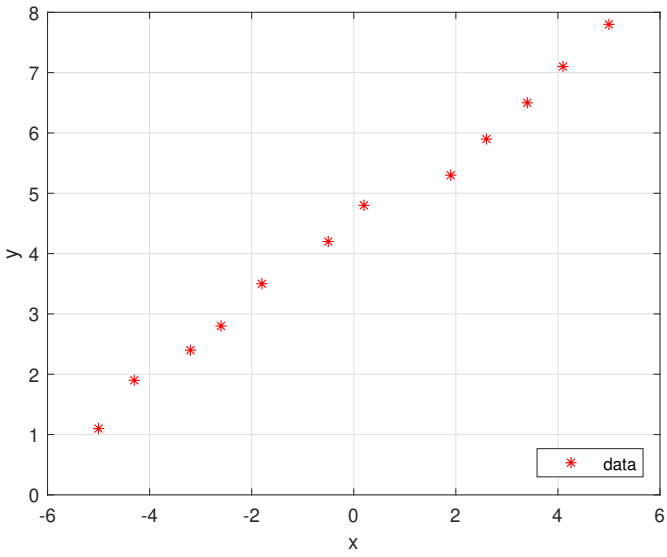

Figure 4.1: Data set representing the position of agents

The local optimization problem is evaluated for each agent independently in the form

$$\begin{aligned} \min_{a_{i,1},a_{i,2}} \quad & (a_{i,2}x_i + a_{i,1} - y_i)^2 \\ s.t. \quad & a_{i,1} = \tilde{a}_1 \\ & a_{i,2} = \tilde{a}_2, \end{aligned} \tag{4.2}$$

where i is the identifier for each agent and N is the number of agents in networked system, $a_{i,1}, a_{i,2}$ are local optimized variable unique for each agent and $\tilde{a_1}, \tilde{a_2}$ represents the global optimum. This problem definition is the same for ADMM and ALADIN, but the global optimization differs and will be discussed in the next part.

4.1.1 Algorithms

The distributed algorithm for the data fitting problem consists of three steps, which are performed sequentially and repeatedly. The algorithm terminates, when each agent converges to the same global result, which means that each agent disposes of the same value of optimized variables.

4.1.1.1 Local Optimization

In the local optimization, the augmented Lagrangian is applied to problem (4.2) in the form

$$L_\rho^i(a_i, \tilde{a}, \lambda_i) = (X_i a_i - Y_i)^2 + \lambda_i^\top (a_i - \tilde{a}) + \frac{\rho}{2}||a_i - \tilde{a}||_2^2, \tag{4.3}$$

where i is the agent identifier, $a_i = (a_{i,1}, a_{i,2})^\top$ is a vector of local optimized variables, $\tilde{a} = (\tilde{a_1}, \tilde{a_2})^\top$ are global optimized variables, $X_i = (x_i, 1)$ and $Y_i = y_i$ represents the private dataset and $\lambda_i^\top$ is vector of Lagrange multipliers.

The local optimal values are then evaluated es follows

$$a_i^\star = \arg\min_{a_i} L_\rho^i(a_i, \tilde{a}, \lambda_i), \tag{4.4}$$

where $\tilde{a}$ and λ_i are constants, with values calculated in the previous iteration by global optimization and dual variable update steps. The a-minimization step is evaluated in consideration of (4.4) for each agent independently and these results are shared among network for global optimization step.

4.1.1.2 Global Optimization

The global optimization, $\tilde{a}$-minimization step, running among all agents, and its computation depends on the type of method.

1. ADMM METHOD

   It uses the simplified version of (2.19b) and it is defined as

$$\begin{pmatrix} \tilde{a}_1 \\ \tilde{a}_2 \end{pmatrix} = \frac{1}{N} \begin{pmatrix} a^\star_{1,1} + a^\star_{2,1} + \cdots + a^\star_{N,1} \\ a^\star_{1,2} + a^\star_{2,2} + \cdots + a^\star_{N,2} \end{pmatrix}, \tag{4.5}$$

   and the result is sent back to the agents.

2. ALADIN METHOD

   The output of global optimization in ALADIN is different. There is a vector of global values, which is distributed to the agents, but its calculated in another way, based on the result from minimization in form

$$\begin{aligned} \min_{\Delta a} \quad & \sum_{i=1}^{N} \left(\frac{1}{2} \Delta a_i^\top H_i \Delta a_i + g_i^\top \Delta a_i \right) \\ s.t. \quad & a^\star + \Delta a = \hat{a}^\star + \Delta \hat{a}, \end{aligned} \tag{4.6}$$

   where $a^\star = (a^\star_1, a^\star_2, \ldots, a^\star_N)^\top$ is the vector of optimal results from agents with $a^\star_i = (a^\star_{i,1}, a^\star_{i,2})^\top$ and $\hat{a}^\star = (a^\star_2, \ldots, a^\star_N, a^\star_1)^\top$. The same applies for increment $\Delta a = (\Delta a_1, \Delta a_2, \ldots, \Delta a_N)^\top$ and $\Delta \hat{a} = (\Delta a_2, \ldots, \Delta a_N, \Delta a_1)^\top$. This optimization problem is also solved by using the Lagrangian form and the global result is then

$$\tilde{a}_i = a^\star_i + \Delta a^\star_i. \tag{4.7}$$

   Here we must be careful about the choice of H and g. We could evaluate them as the Hessian matrix and gradient, but only for a task, where data privacy does not play a role. In problems concerning the privacy, H and g have to be evaluated differently. In our case, they have been initialized randomly and tuned to provide a satisfactory result.

4.1.1.3 Dual Variable Update

This step is executed again by each agent with its private data information. So the dual variable update has a form

$$\lambda_i^{k+1} = \lambda_i^k + \rho(a_i - \tilde{a}). \tag{4.8}$$

The algorithm is repeated until the stopping criterion is satisfied. For specific details of ADMM or ALADIN algorithm, see Algorithm 1 or Algorithm 2 respectively.

4.1.2 Results for Data Fitting

The convergence of methods depends on intial values of optimized variables $\tilde{a}_0$, the value of parameter ρ and value of tolerance for stopping criteria ε. The initial points $\tilde{a}_0 = (0, 0)$, tolerance $\varepsilon = 10^{-3}$ and $\rho = 10$ have been specified.

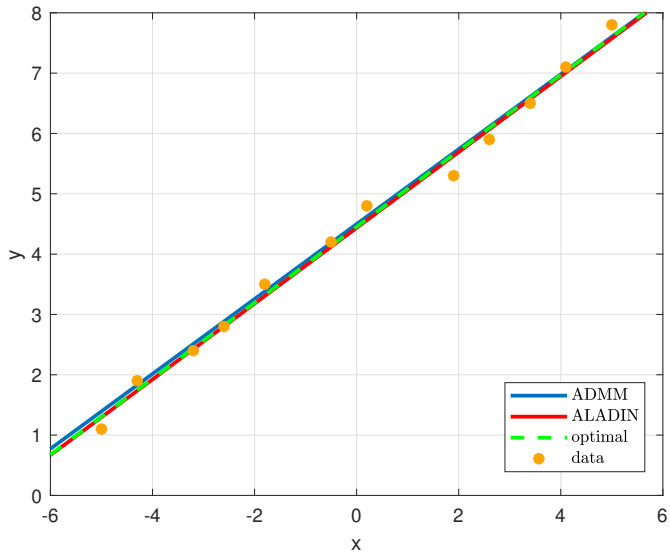

Figure 4.2: Comparison of ADMM and ALADIN for data fitting problem

As is shown on Fig. 4.2, we can compare methods by precision of approximation. The value of cost function in optimum for ADMM is $f(a^\star_{\mathrm{ADMM}}) = 0.3982$ in comparison with the value $f(a^\star_{\mathrm{ALADIN}}) = 0.3674$. There is also difference in the number of iterations needed for the methods to convergate nearby the global optimum of the original problem. Iterations needed for ADMM $k_{\mathrm{ADMM}} = 23$ are incomparably to the interations of ALADIN method $k_{\mathrm{ALADIN}} = 6$. The convergence is ilustrated on Fig. 4.3 for ADMM and on Fig. 4.4 for ALADIN, where $a^\star$ represents the global optimum for initial problem (4.1).

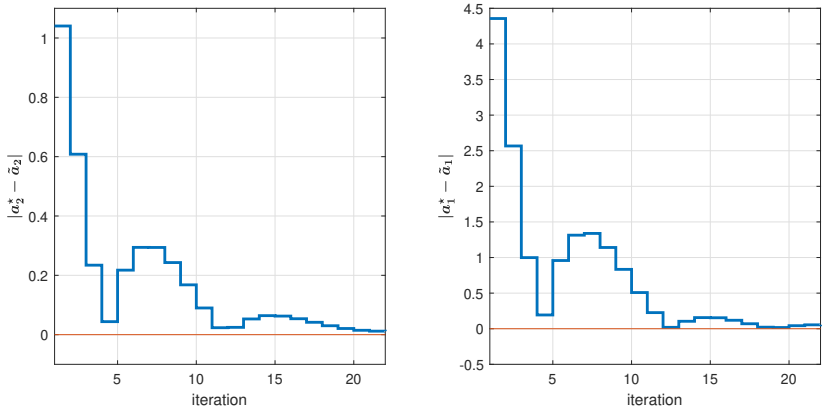

Figure 4.3: Convergence of ADMM

Thse results are also summarized in Tab. 4.1, where errors in optimum e_1 and e_2 are calculated as follows

$$e_1 = \frac{|a_1^\star - \tilde{a}_1|}{a_1^\star}, \quad e_2 = \frac{|a_2^\star - \tilde{a}_2|}{a_2^\star}. \tag{4.9}$$

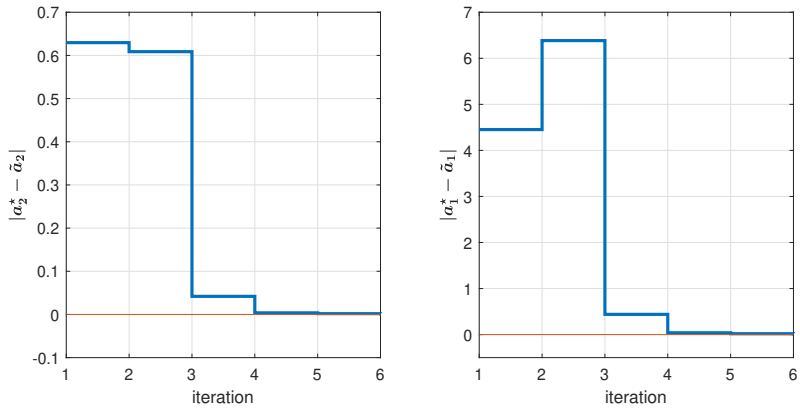

Figure 4.4: Convergence of ALADIN

Table 4.1: Comparison of ADMM and ALADIN method applied to data fit problem.

Method	Objective value $f(a^\star)$	No. of iterations	Error in optimum (%)	
			e_1	e_2
ADMM	0.3982	23	1.0	1.4
ALADIN	0.3674	6	0.4	0.3

As a result, we assume the ALADIN as a more effective method for solving data fitting problem according to the number of iterations and precision of the minimization.

4.1.3 Centralized vs. Decentralized Approach

If we look at the algorithms introduced for this regression problem, we will understand, it is assumed that there is some central unit, which provides global optimization. The general idea of presenting the results above of a centralized approach is to prove the functionality of algorithms before we will dive into a more complex decentralized approach.

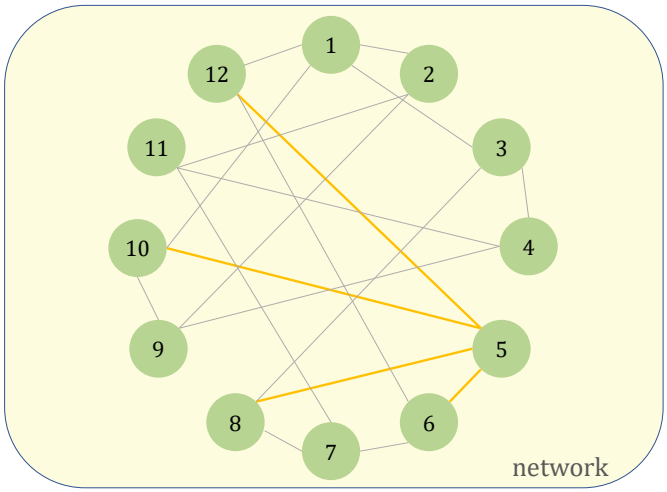

Figure 4.5: Structure of network with connections in decentralized approach

We have already discussed the advantages and disadvantages of both types of optimizations in Sec. 2.6. When it comes to algorithms, the only difference is in the global optimization step. It is no longer evaluated in some central unit, but each agent computes their own *global* optimal variables, based on information from agents, with which it is interconnected. In this comparison, the convergence rate of 5-th agent is shown. Fig. 4.5 illustrates the connection between agents in the network, where the central unit is absent. The connections of monitored 5-th agency are highlighted with orange color.

4.1.3.1 ADMM

Let's see the comparison of centralized and decentralized optimization provided by the ADMM method. Even though this agent has only 4 connections, its convergence properties are comparable to the properties of a centralized approach (Fig. 4.6 and

Fig. 4.7).

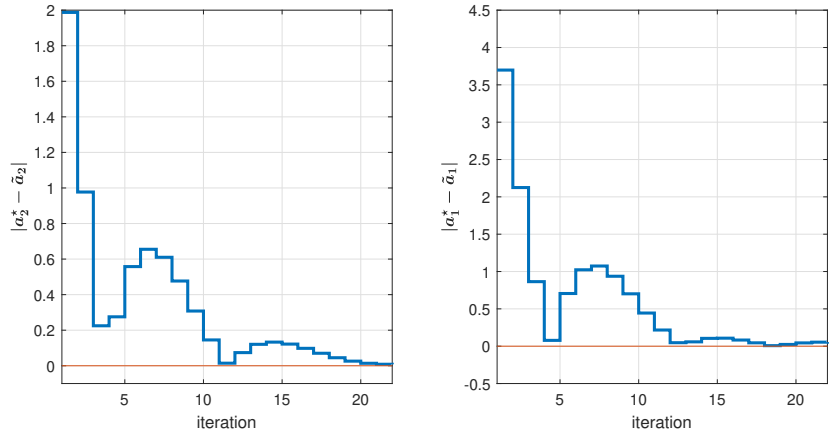

Figure 4.6: ADMM convergance rate of 5-th agent in centralized approach

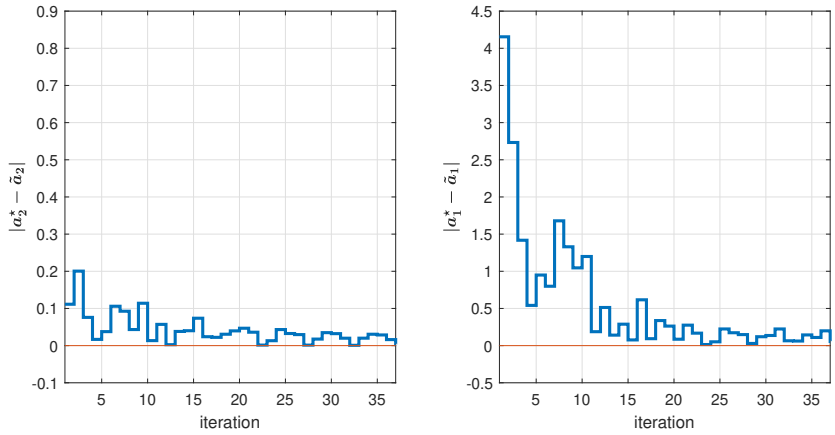

Figure 4.7: ADMM convergance rate of 5-th agent in decentralized approach

The number of needed iterations for the decentralized ADMM method has increased from 23 to 37 in comparison with a centralized algorithm. The accuracy of the result (Tab. 4.2) is worse, but the difference is negligible.

Table 4.2: Comparison of centralized and decentralized approach of ADMM method on data fitting problem

Approach	Objective value $f(a^\star)$	No. of iterations	Error in optimum (%)	
			e_1	e_2
Centralized	0.4012	23	1.0	1.3
Decentralized	0.4080	37	1.2	1.3

The more interesting thing is the increasing tendency to oscillate. It is caused by indirect information exchange. Although the 5-th agent is not connected to each agent, its neighbors are connected to other agents, so the result provided by them indirectly carries information about all local optimized variables. In the end, the algorithm converges to the global solution, due to this structure. It may seem that the algorithm should have ended much sooner, but we need to keep in mind, there are eleven other agents and the algorithm terminates when each agent will satisfy the stopping criteria.

4.1.3.2 ALADIN

Now, we will compare the centralized and decentralized approach for a more complex ALADIN method.

Table 4.3: Comparison of centralized and decentralized approach of ALADIN method on data fitting problem

Approach	Objective value $f(a^\star)$	No. of iterations	Error in optimum (%)	
			e_1	e_2
Centralized	0.3982	6	1.0	1.3
Decentralized	0.4218	21	1.3	1.4

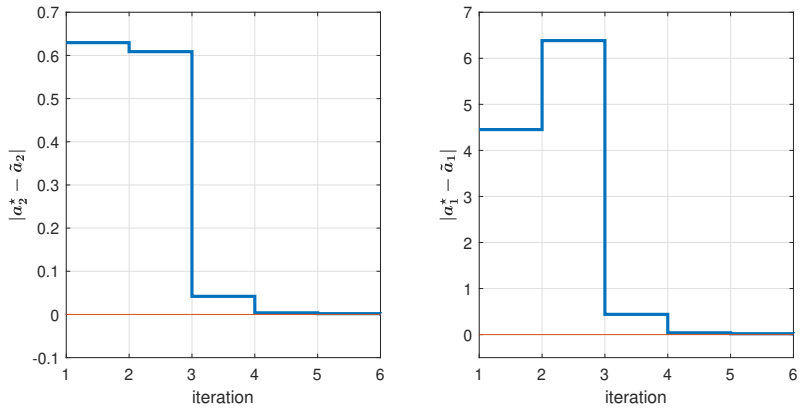

Figure 4.8: ALADIN convergance rate of 5-th agent in centralized approach

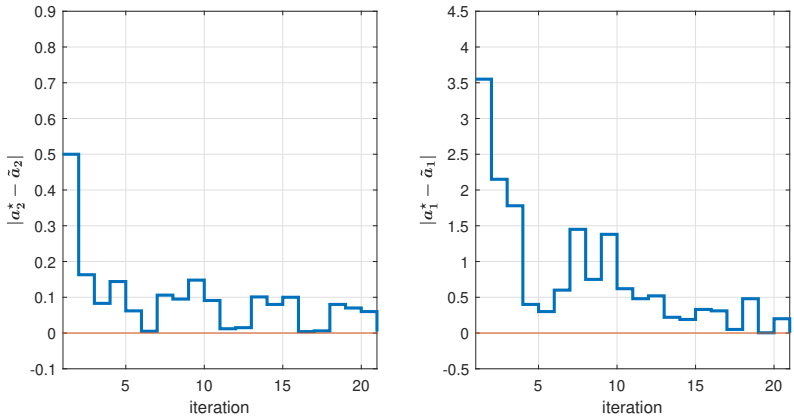

Figure 4.9: ALADIN convergance rate of 5-th agent in decentralized approach

It is obvious, that the decentralized approach is significantly worse in comparison with centralized. It takes 15 iterations more, and the result is not that precise as in centralized (see Tab 4.3). But it still converges very well. If we compare the number of iteration in ADMM and ALADIN decentralized optimization, ALADIN still provides a better result in the convergence rate.

4.2 Event Scheduling

Many times it happens that we meet with the problem when it is necessary to plan a meeting to suit all participants. For the purpose of this thesis, the term event scheduling refers to searching for a time slot that is empty for each participant's calendar. We assume three work calendars as shown in Fig. 4.10. The first issue of this task is the non-smoothness of a cost function.

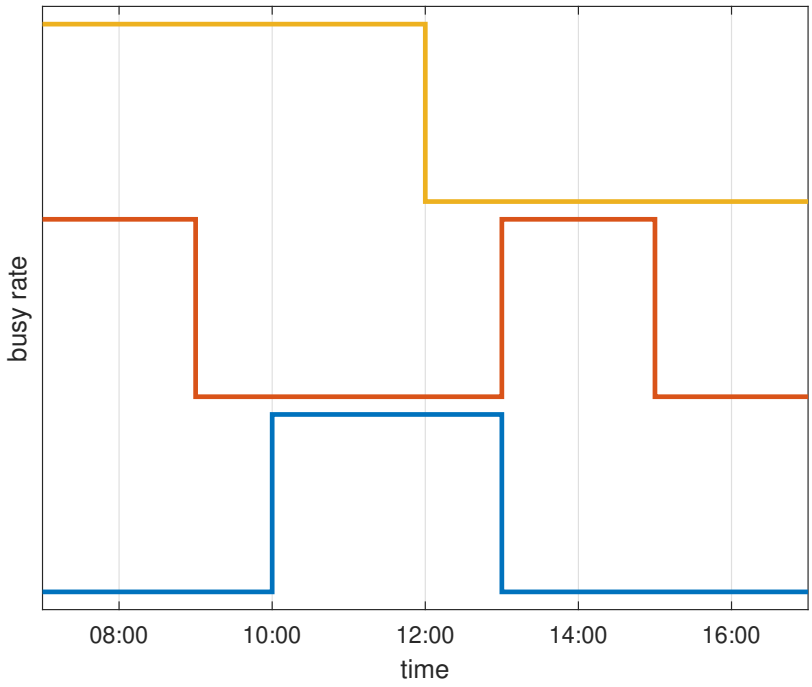

Figure 4.10: Ilustration of the three work calendars

We decided to approximate the functions representing the calendar by a higher-degree polynomial to achieve a smooth function. The most precise approximation of these

functions was provided using the 6th-degree polynomial in the form

$$P = p_6x^6 + p_5x^5 + p_4x^4 + p_3x^3 + p_2x^2 + p_1x + p_0, \tag{4.10}$$

where $p = (p_0, \ldots, p_6)$ are coefficients of polynomial, which are calculated using optimization and x is vector of data representing the calendar. The aproximation is shown on Fig. 4.11.

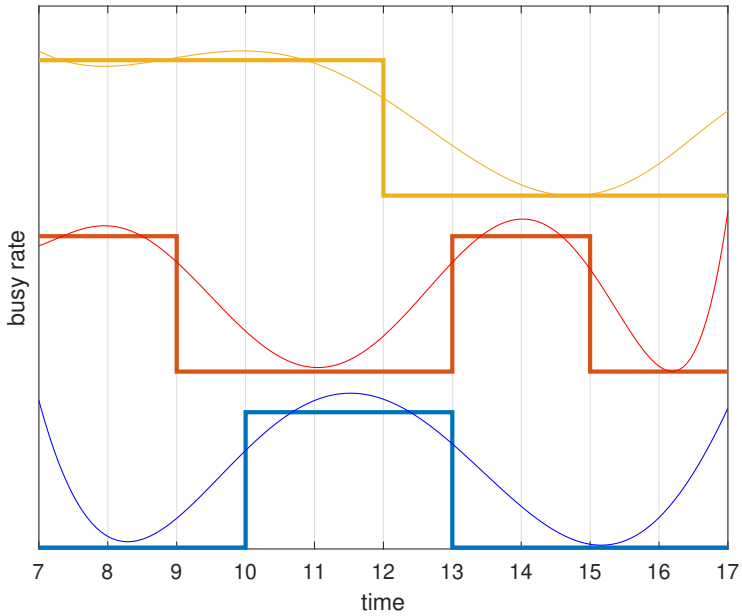

Figure 4.11: Aproximation of calendar functions

As we can see, this problem is nonconvex, which signifies the second issue of calendar optimization. In general, ALADIN is designed to solve even nonconvex optimization problems, but it was proven that ADMM may be divergent if $f(x)$ is nonconvex [10]. However, using ADMM on this particular problem (event scheduling) leads to a satisfactory result. In this particular problem, the term satisfactory result signifies, that each agent has found the same half-hour block for scheduling the event.

4.2.1 Algorithm

The distributed optimization algorithms for event scheduling is very similar to the algorithms for data fitting problem. They consist of the same three parts: local optimization, global optimization, and dual variable update. The original problem of event scheduling is mathematically defined as

$$\min_x f(x) = \min_x \sum_{i=1}^{N} f_i(x) = \min_x \sum_{i=1}^{N} P_i(x). \tag{4.11}$$

Since we have different cost functions, the only difference between data fitting problem and event scheduling problem is in the formulation of augmented Lagrangian, used in the local optimization step. It has the following form

$$L_\rho^i(x_i, \tilde{x}, \lambda_i) = P_i(x_i) + \lambda_i^\top (x_i - \tilde{x}) + \frac{\rho}{2}||x_i - \tilde{x}||_2^2, \tag{4.12}$$

where x_i represents the optimized variable (in this case it is a time) of i-th calendar, and P_i is the polynomial representing the participants availability. The local optimal time for organizing a meeting is computed as

$$x_i^\star = \arg\min_{x_i} L_\rho^i(x_i, \tilde{x}, \lambda_i). \tag{4.13}$$

The global and dual variable update steps are then evaluated as in the data fitting problem, depending on which method is applied. For ADMM the global optimization step is evaluated as (4.5), for ALADIN as (4.6) and the update of the Langrangian multiplier as (4.8).

4.2.2 Results for Event Scheduling

The setup for this problem was the same as for the data fitting problem, but tolerance has to be adjusted, because the $\varepsilon = 0.08$, which represents the precision of 5 minutes, was enough to achieve a good solution. The important thing to mention, the local optimization was performed by numerical optimization methods (gradient method, Luus-Jaakola method, etc.), so there is a high dependence on the initial point.

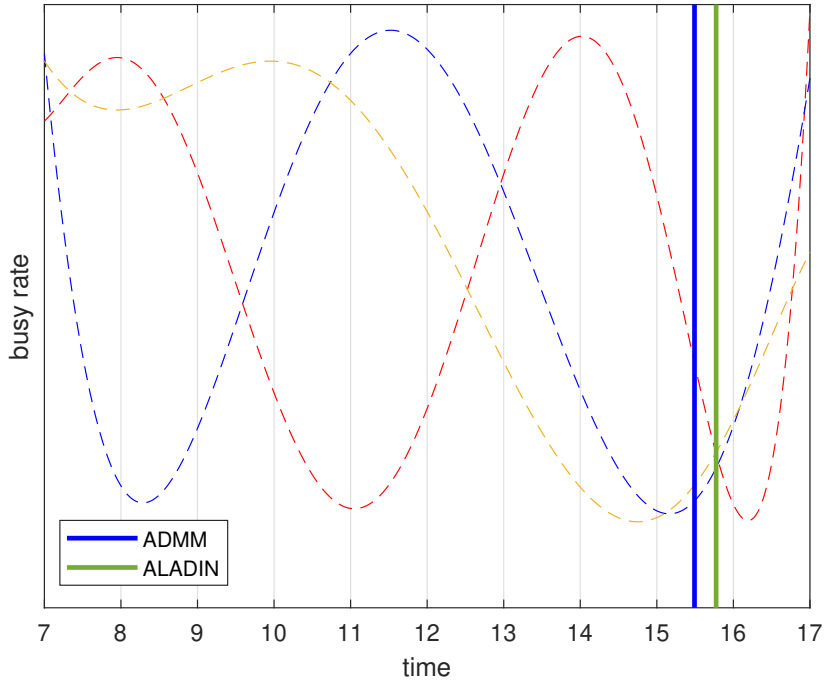

Figure 4.12: Solution of optimization for ADMM (blue line) and ALADIN (green line)

As we can see on Fig. 4.12 the ALADIN methods provided more precise result then ADMM. Summarization is in Tab. 4.4.

Table 4.4: Comparison of centralized and decentralized approach of ALADIN method on data fitting problem

Method	Objective value $f(x^\star)$	No. of iterations	$x^\star$
ADMM	5.0140	20	15.53
ALADIN	4.4397	9	15.77

Specifficaly, the value of cost function $f(x^{\star}_{\text{ALADIN}}) = 4.4397$ for ALADIN result $x^{\star}_{\text{ALADIN}} = 15.77$ turned out to be smaller then $f(x^{\star}_{\text{ADMM}}) = 5.014$ for ADMM result $x^{\star}_{\text{ADMM}} = 15.53$.

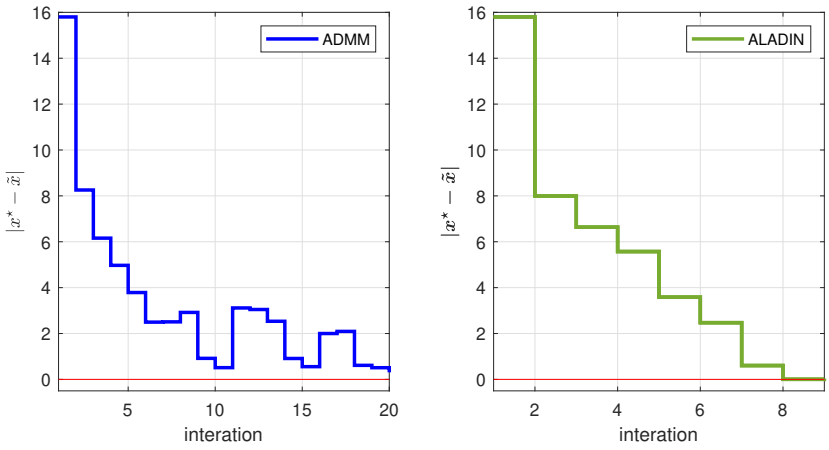

Figure 4.13: Convergence rate for ADMM (blue line) and ALADIN (green line) algorithms

The amount of iterations needed for convergence confirms the ALADIN method $k_{\text{ALADIN}} = 9$ as more convenient for data fitting problem then ADMM $k_{\text{ADMM}} = 20$ (Fig. 4.13). The reason can be found in different global optimization step.

4.2.3 Centralized vs. Decentralized Approach

To compare the centralized and decentralized approach for event scheduling, we decide to add more participants with their calendars, because the application of decentralized optimization on the problem with three agents would not be very justified. That is the reason, why we added two more agents to the system. So now, we are searching for an empty spot among five calendars, as shown in Fig. 4.14. As we can see, the block, where each agent is free, is between 9:00 am and 10:00 am.

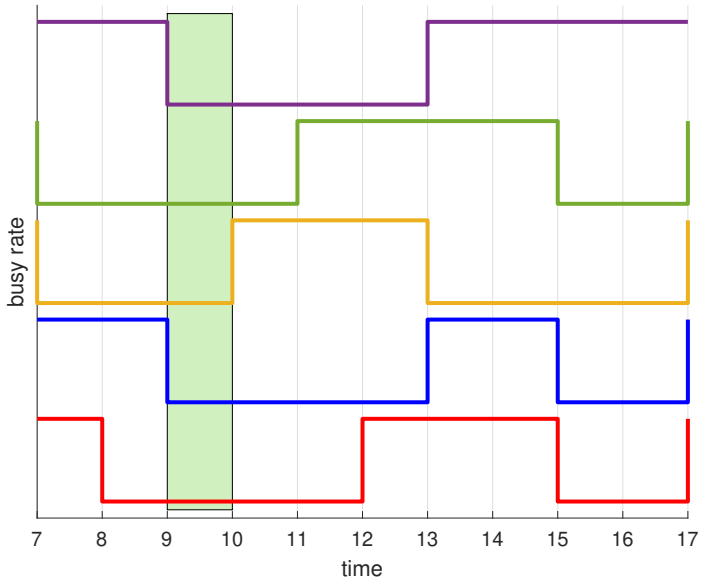

Figure 4.14: Illustration of all calendars with a common optimal time

The results of decentralized optimization algorithms differs based on defined connections between agents. We decided to use the structure of the network as shown in Fig. 4.15.

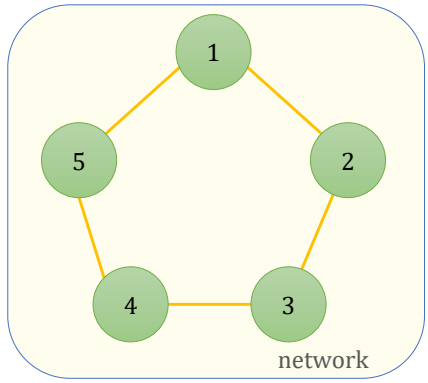

Figure 4.15: Structure of network with connections in decentralized approach

To illustrate the convergence properties of a centralized and decentralized approach for ADMM and also ALADIN method, we decided to show the result for only one agent. It is not important, which agent is chosen because the algorithm terminates only when each agent has a similar value of the optimized variable.

4.2.3.1 ADMM

After applying the ADMM method in both ways, it is evident, that the convergence rate of a centralized approach is faster (Fig. 4.16) then in decentralized, although the number of iterations, 35 for centralized and 41 for decentralized, is not very contrasting.

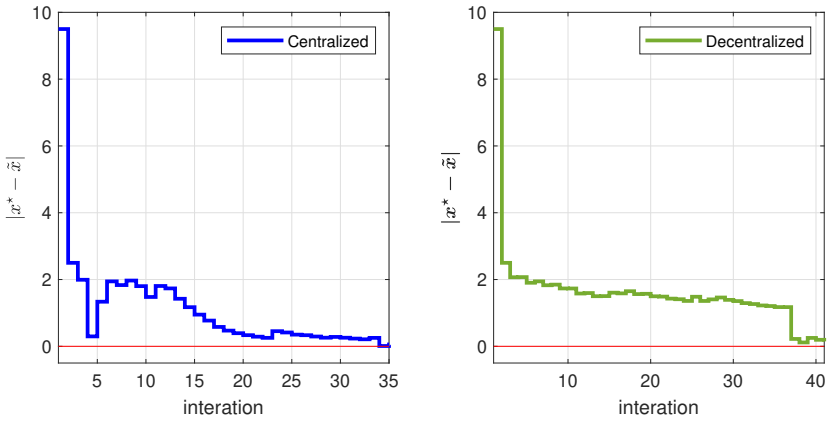

Figure 4.16: ADMM convergance rate of 3-th agent in centralized and decentralized approach

In terms of accuracy, both techniques have achieved excellent results. The values of the optimal time and the objective function is located in Tab. 4.5. The decentralized optimization process is a little bit slower, as expected, but the result is still very precise.

Table 4.5: Comparison of centralized and decentralized approach of ADMM method on event scheduling problem

Approach	Objective value $f(x^\star)$	No. of iterations	$x^\star$
Centralized	9.9190	35	9.2910
Decentralized	9.9610	41	9.3507

These results are encouraging since the goal is to replace the centralized approach with decentralized approach, so the whole procedure of searching the optimal variable can be distributed to agents without any central unit involved.

4.2.3.2 ALADIN

In general, the convergence of the ALADIN method is faster, in comparison with ADMM, regardless of the approach. Regarding the number of iterations and the accuracy of the centralized and the decentralized algorithms, the results are almost indistinguishable. Tab. 4.6 provides a summary of these values.

Table 4.6: Comparison of centralized and decentralized approach of ALADIN method on event scheduling problem

Approach	Objective value $f(x^\star)$	No. of iterations	$x^\star$
Centralized	9.9433	16	9.3248
Decentralized	10.2400	20	9.4123

A more interesting thing can be seen in Fig. 4.17, where the convergence rate has no gradually decreasing character. The explanation can be found in Fig. 4.14. In this

case, the 3-th agent is used to present the results (yellow calendar). This agent is interconnected with two other agents (green and blue calendars). The first iterations of an algorithm provide the optimal time as 16:00. It is not an incorrect behavior, because it is a local optimum for mentioned three agents. The further the algorithm runs, the value of an optimized variable is affected by remaining calendars, which causes convergence to the global optimum.

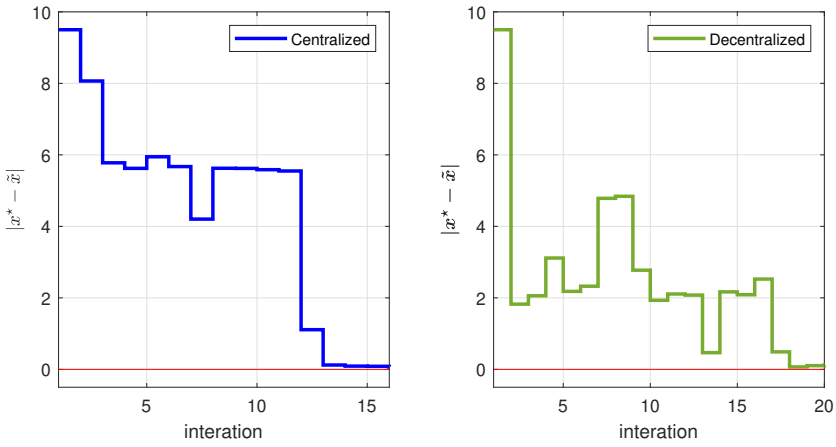

Figure 4.17: ALADIN convergance rate of 3-th agent in centralized and decentralized approach

In this particular comparison is shown, that ADMM provides smoother change in convergence rate than ALADIN. Since ADMM is based on averaging the optimal variables, convergence error decreases slowly without huge osciallations. On the other hand, in ALADIN method the result of global optimization problem is calculated, which leads to big changes in convergence rate, but also to faster termination.

Chapter 5

Decentralized Neural Network Training

In this chapter, we will declare the results of the proposed distributed and decentralized artificial neural network training. Several special cases are introduced, in order to test the robustness of the learning process. The structure of a neural network, which was used in test examples is illustrated in Fig. 5.1.

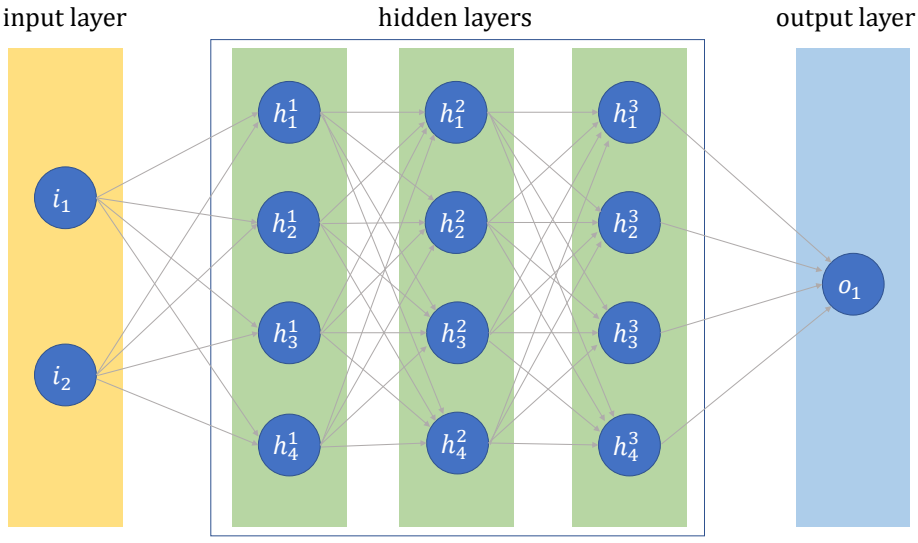

Figure 5.1: Structure of the neural network used for the training tests

A chosen neural network contains of

- 1 input layer with 2 neurons,
- 3 hidden layers with 4 neurons at each layer,
- 1 output layer with 1 neuron.

In order to provide all test trainings, the Deep Learning Toolbox build in MATLAB is used. We need to supply this toolbox with information about the structure of the neural network mention above, the input and target dataset and we can also initialize the weights and biases. In MATLAB, three different data structures have to be defined.

$$IW = \Big\{\{4 \times 2 \text{ double}\}\Big\}, \quad LW = \begin{Bmatrix} \{4 \times 4 \text{ double}\} \\ \{4 \times 4 \text{ double}\} \\ \{4 \times 1 \text{ double}\} \end{Bmatrix}, \quad b = \begin{Bmatrix} \{4 \times 1 \text{ double}\} \\ \{4 \times 1 \text{ double}\} \\ \{4 \times 1 \text{ double}\} \\ \{1 \times 1 \text{ double}\} \end{Bmatrix}.$$

The input layer weight structure IW with dimension 4×2, since there are 8 connections between input neurons and neurons in the first hidden layer (Fig. 5.1). The hidden layer weight LW contains of 36 values, 16 between each connected hidden layers and 4 between last hidden layer and output layer. Finally, there are 13 values in a bias matrix b, 12 for neurons in all hidden layers and 1 for output neuron.

5.1 Decentralized Training

We consider the urgency to teach a neural network to control the simple system (movement of a mass point). The system has the following form

$$\dot{x} = Ax + Bu = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \tag{5.1}$$

where A is dynamic matrix, B is input matrix, $x = [x_1, x_2]^\top$ is a vector of two states and u is control input.

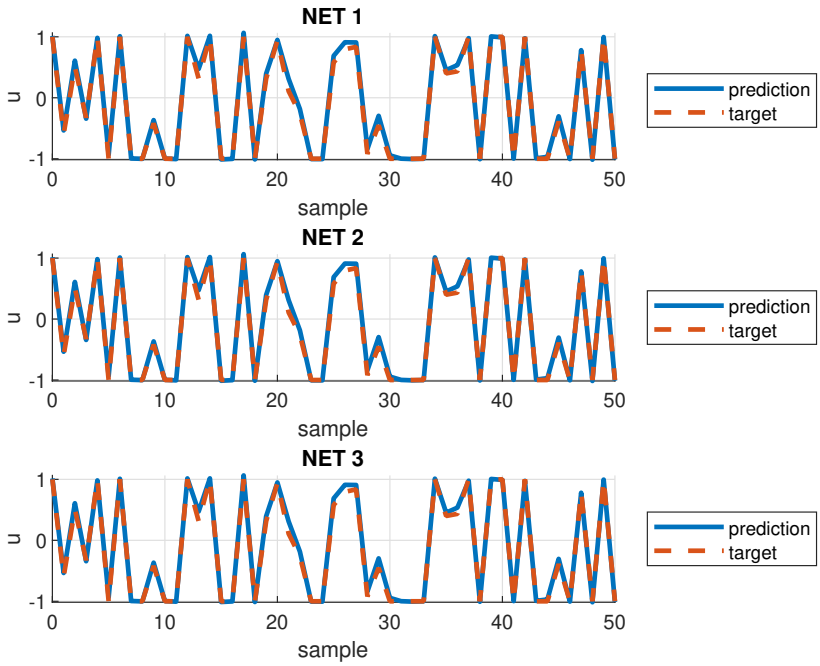

Figure 5.2: Comparison of predicted and target outputs of 3 trained networks

The data for the neural network training is available and the size of this dataset is $[1850 \times 3]$. The aim is to train this a neural network in distributed way. So the dataset is split into four sub-sets. The three of them are used for training three different neural networks. Their parameters are shared between each other and the average of them is computed in every single step. The convergence is reached iteratively. In the end, there are three neural networks, with the same parameters. So in general, we can say, there is one neural network, trained to control the given system. The last data sub-set is used to verify the quality of the trained neural network.

As we can see in Fig. 5.2, all sub-nets are trained very precise and provide the expected result in comparison with test data (the last sub-set).

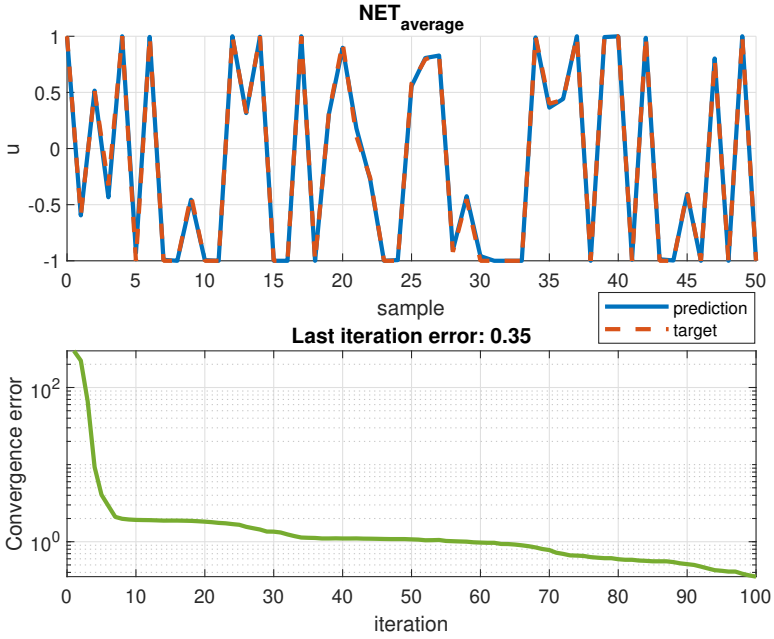

Figure 5.3: The properties of average neural network

The convergence error (CE) is computed as follows

$$CE = \sum_{i=1}^{N_s} (\hat{u}_i - u_i)^2, \tag{5.2}$$

where N_s is the number of testing samples, $\hat{u}$ is the output provided by the trained network (prediction) and u is the expected output (target) defined in the last sub-set. The bias, input weight, and layer weight are structures with dimensions defined at the beginning of this chapter. The parameter error (PE) of mentioned structures (as shown on Fig. 5.4) is defined as

$$PE = \sum_{i=1}^{N_p} |\overline{p}_i - p_i|, \tag{5.3}$$

where N_p is the number of parameters in structures, p is the value of an exact parameter in structure and $\overline{p}$ is the average value of this parameter calculated as

$$\overline{p} = \frac{1}{N_N} \sum_{i=1}^{N_N} p_i, \tag{5.4}$$

with N_N defined as the number of trained neural networks.

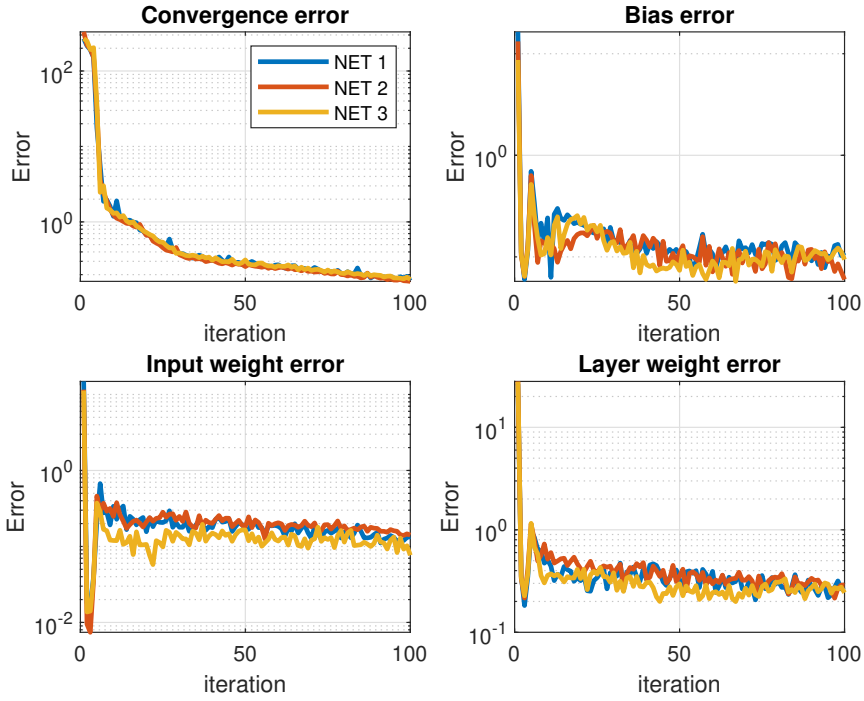

Figure 5.4: The CE and PE values in each iteration

As we can see on Fig. 5.4 and in Tab. 5.1, the convergence error is very small and the convergence is very fast. $CE = 1$ is already reached in 10th iteration. The number of (training) iterations is set to $k_{max} = 100$.

Table 5.1: The errors of decentralized training of neural networks in last iteration

Neural net	Convergence error	Bias error	Input weight error	Layer weight error
Net 1	0.165	0.111	0.134	0.258
Net 2	0.163	0.060	0.141	0.288
Net 3	0.173	0.094	0.077	0.243

When the training part is over, the average neural network is obtained and it is used to control the given system, even it was not trained at all. But its parameters are gained as average values from parameters of trained networks. The final properties, the precision, and convergence of the average neural network, are summarized in Fig. 5.3, where $CE = 0.35$.

5.1.1 Convergence Comparison of Training with N Agents

In this part, we will try to answer the following question. What happens with convergence rate, if the initial dataset is decomposed to more than three agents? The first guess would be, that if we have more agents, each of them operates with a smaller amount of data, naturally, the convergence rate will be slower. The answer is in Tab. 5.2 based on Fig. 5.5.

Table 5.2: Comparison of training with different number of agents

No. of agents	Convergence error	Iteration	
		CE = 1	CE = 0.5
10	0.153	10	20
20	0.171	10	35
50	0.914	50	-
100	1.897	-	-

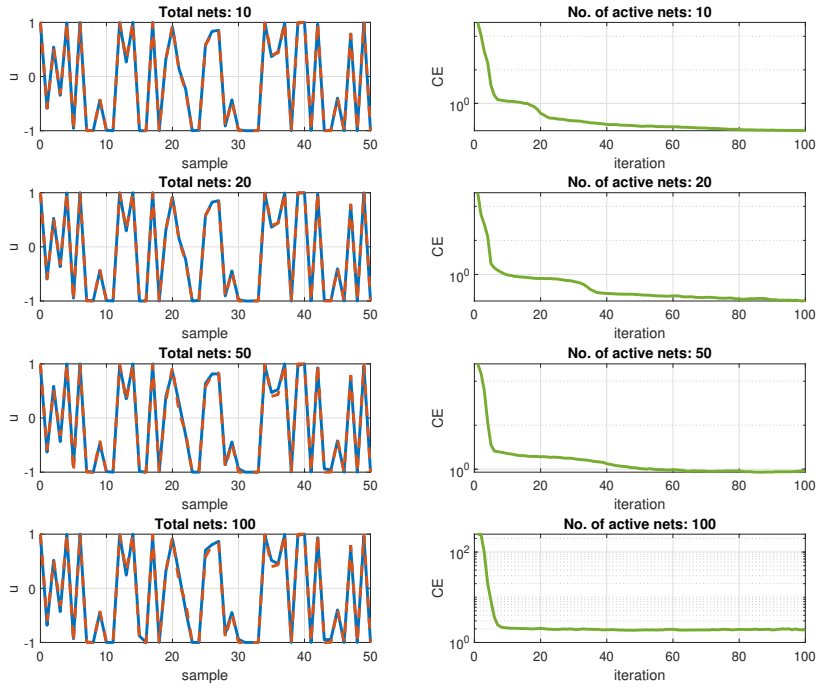

Figure 5.5: Comparison of precision and convergence rate of training with different number of agents in network

It is obvious that our assumption was correct. The convergence error increases with a higher number of agents. In parallel, the convergence rate is different. While the cases with 10 and 20 agents attain the $CE = 1$ in 10th iteration, the case with 50 agents needs 50 iterations and the case with 100 agents will never reach mention value. If we compare the first and second cases, their convergence rate is quite similar at the beginning. But it changes after several iterations. The convergence rate of 10 agents is faster because $CE = 0.5$ is achieved after 20 iterations, while the second case needs 35 iterations.

5.1.2 Training with Nonuniform Distribution of Data

A very important thing for neural network training is to test special cases. The term special case signifies the situation, which is not commonly expected, but there is no guaranty, it will not appear. One of the special cases is the non-uniform distribution of data to the agents. It means, that each agent (for this example three agents were used) gets the dataset with the different output region (see Fig. 5.6).

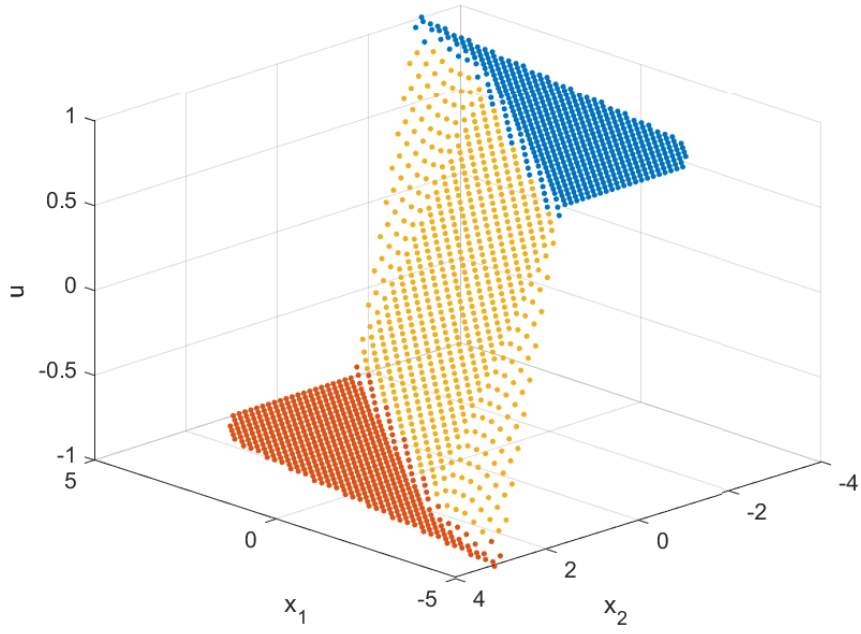

Figure 5.6: Nonuniform distribution of data (red region - 1. agent, blue region - 2. agent, yellow region - 3. agent)

Even the agents have different training output regions, the output of the trained (average) network is very precise. The convergence rate is not that fast as in previous examples, which means the more iteration is needed to converge (almost 90), but it is

satisfactory since the convergence error is equal to 0.28. For the better visualization of results and convergence graph see Fig. 5.7.

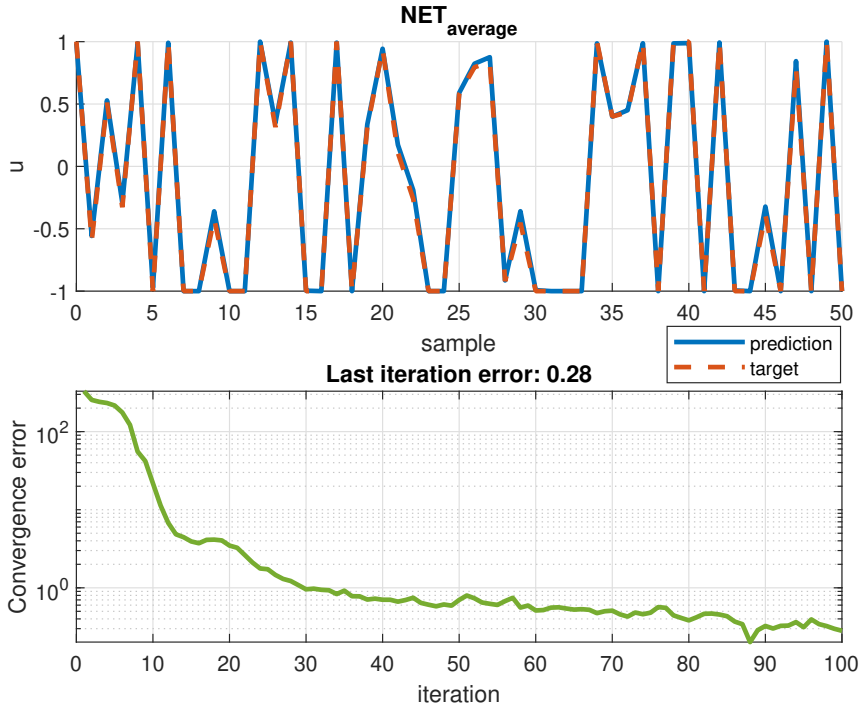

Figure 5.7: Average network for nonuniform data distribution case

So, even each agent manipulates with completely different dataset concerning the same problem, a common solution can be found, by sharing the partial results.

5.1.3 Centralized vs. Decentralized Training

As discussed before, the centralized and decentralized optimization has several pros and cons. The same applies to both types of trainings. But how it differs in convergence and precision properties?

In centralized training, we assume all agents train in each iteration, and the parameters are shared centrally. So the average parameters are computed based on parameters from every agent. On the contrary, decentralized way point to the situation, where each agent evaluates the average on its own in pursuance of the parameter values provided by agents in the network to which it is connected.

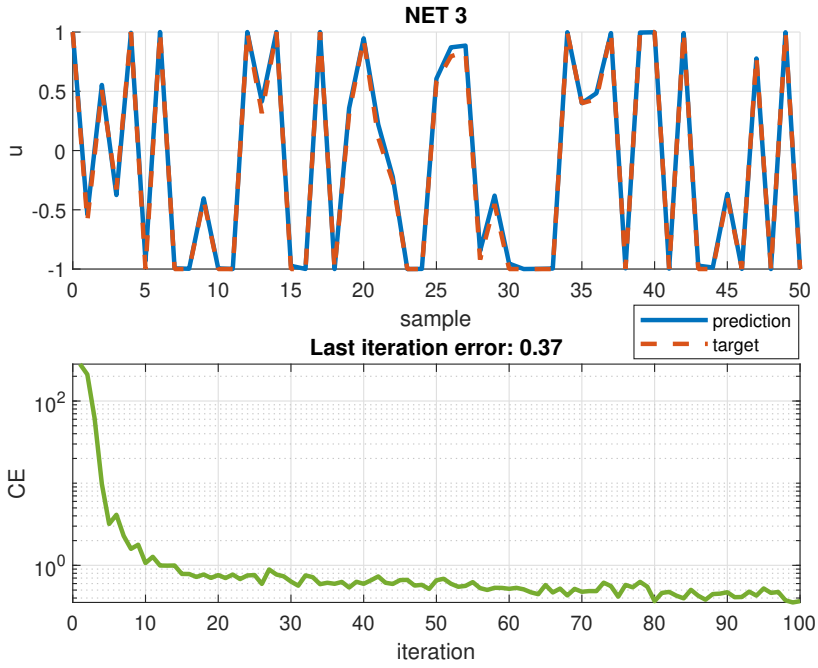

Figure 5.8: Result of centralized neural network training

So we performed a simple scenario. Ten agents are used to train a neural network. Fig. 5.8 illustrates the centralized way of training, meaning that all 10 agents train at the same time and create one central average net. The convergence rate is very fast. After 20th iteration, there is no significant change in CE. The last iteration error is $CE = 0.37$.

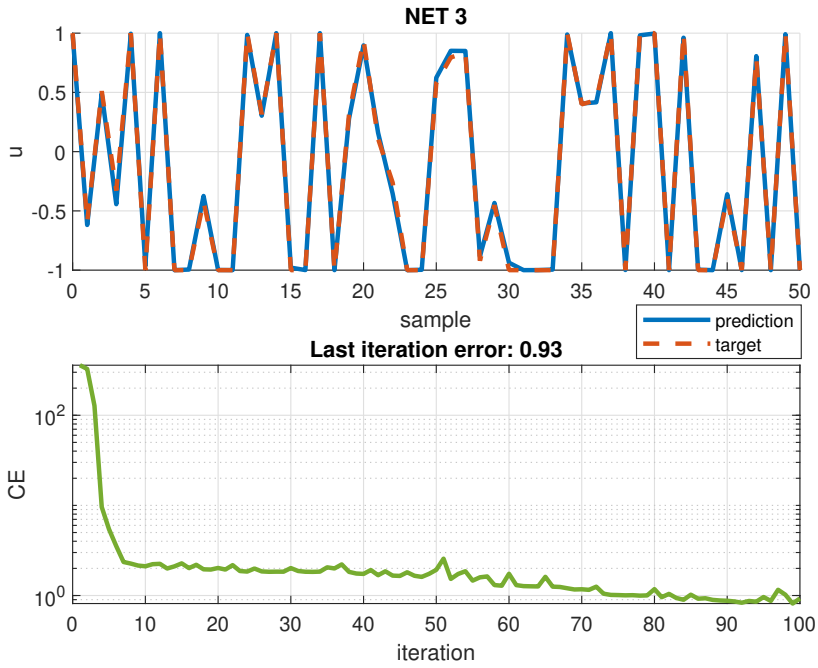

Figure 5.9: Result of decentralized neural network training

On the other hand, Fig. 5.9 shows the decentralized training, where convergence rate is much worse, slower, the number of needed iterations to converge is around 90, but the last iteration error $CE = 0.93$ is tolerable. The explanation of the degraded quality of the learning process is, that each agent is connected only with two more agents, so the information flow is reduced in comparison with a centralized way.

5.1.4 Dynamical Change in Active Agents

The last test case is oriented on the possibility to train the network without having to wait to receive data from all connected calculation units. So, we consider 10 agents in network and only several of them contribute to the average computation. The number of contributors called active agents is randomly generated in each iteration. The goal

is to simulate the behavior of real devices, where their computational speed can differ, there is some data transfer delay, the calculation stops working or it refuses further cooperation.

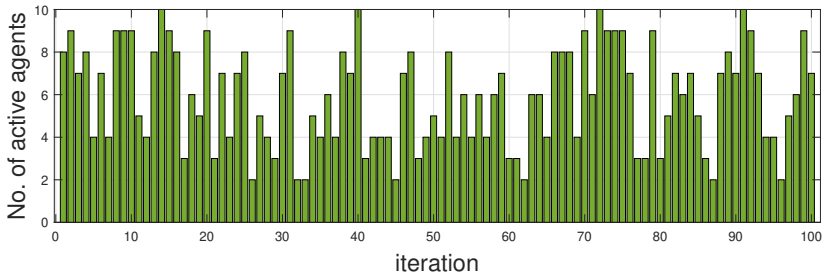

Figure 5.10: Number of active nets per iteration

Fig. 5.11 illustrates the number of iterations (corresponding iteration count) belonging to a given number of active agents. The general distribution over the whole simulation is shown in Fig. 5.10, so we can see, how many agents provided their information per iteration.

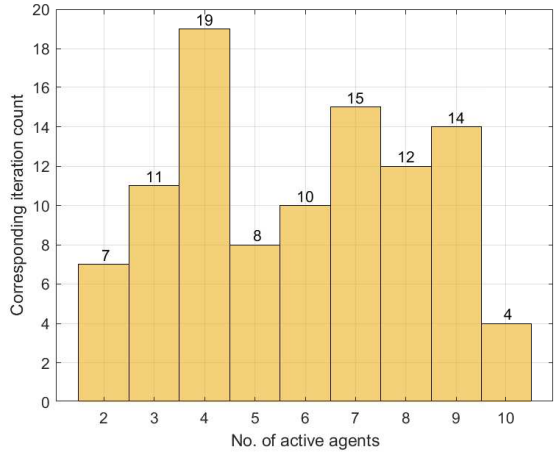

Figure 5.11: Number of active agents corresponding to iteration count

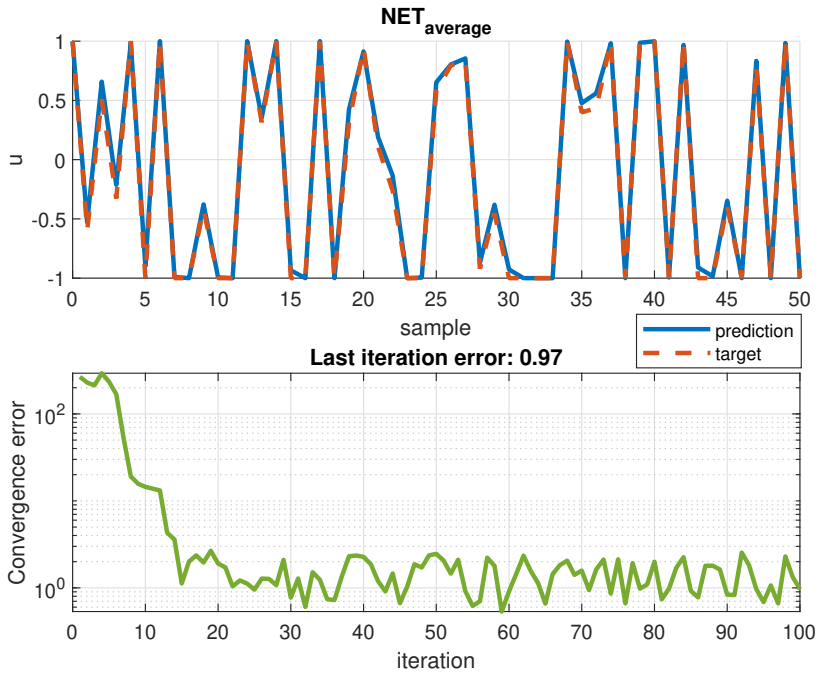

Figure 5.12: Result of neural network training with dynamical change in active agents

The results are shown in Fig. 5.12. We can see, the convergence error starts to oscillate around $CE = 1$ after 20 iterations and there is no improvement. But $CE = 0.97$, for 350 testing samples, is still satisfying. So, even there is a problem with communication or some calculation units do not provide their results, the observed agent can still train its neural network with satisfactory accurancy.

Chapter 6

Conclusions

The aim of this thesis was to design and implement a system, that is able to process optimization tasks and machine learning in a decentralized enviroment with guaranteed privacy of agent's data. The mentioned desgin and implementation were provided on several tasks with satisfactory results as described in experimental part of thesis.

The first step of this thesis was to solve the optimization problem of a networked system, where each agent keeps its private and sensitive data, important for the optimization through the system. Therefore, distribued optimization and the modern algorithms for its solution were introduced. The most commonly used, ADMM and ALADIN, were implemented in MATLAB for cases with convex and also non-convex cost functions. In both cases, ALADIN method came out as preferred, because of a faster convergence rate. This statement is based on the results of the data fitting and event scheduling problem.

Regarding the precision of the solution, both methods were comparatively good. Both algorithms were affected by stopping criteria and mainly by chosen tolerance ε. Tolerance changed according to the optimization problem. The results were also influenced by the choice of the local optimization algorithm. The type of algorithm depends on the convexity of the objective.

In addition, the comparison of a centralized and decentralized approach was realized for

all problems and each method. According to the results, the decentralized optimization approach has a slower convergence rate than centralized. It is caused by indirect data exchange among the network. But the accuracy of the result was almost the same as in a centralized way. This is the positive information, because it means, that we do not need any central unit to supervise the whole optimization process through the network. On the contrary, we can benefit from all advantages of a decentralized approach, such as lower computational burden and memory footprint per calculation unit, elimination of single point of failure, etc.

The understanding of distributed and decentralized optimization was just the first step of the thesis. The main goal was to apply this knowledge to machine learning algorithms. We decided to choose the artificial neural network as a model that can be trained in not only distributed but also a decentralized way. It can be used in emerging Federated learning and other modern applications. The result of our distributed training algorithms was the neural network, which provides very precise result according to MSE parameter. To test the robustness of distributed training, we came up with several special tests, such as the non-uniform distribution of data or dynamical change in active agents. Both converged to the 100-th iteration with the convergence errors, which were considered as sufficient.

Finally, the comparison of centralized and decentralized learning was made. The convergence error of centralized learning was smaller in comparison with decentralized and the same applies for the number of iterations. We expected the slower convergence rate of a decentralized approach, but such a deterioration in accuracy was not predicted. However, the convergence error was calculated as a difference between predicted and target output on 350 samples, so it is still tolerable.

In conclusion, the centralized optimization algorithm can be replaced by decentralized, if the slower convergence rate is not an issue. The choice of distributed optimization method is based on the definition of the problem. Even though the ALADIN method is

preferred in our test problems, the H and g setup for tasks with privacy concerns can be challenging. This is something, that could be further studied to avoid the laborious tuning.

The previous conclusion about replacement of centralized approach by decentralized applies also to machine learning. In addition, in neural network training, we must be careful about the number of agents in the network. According to Sec. 5.1.1, the higher number of agents, the worse convergence, and accuracy properties. But this is true only for cases, where one dataset is divided between a huge amount of agents or the agents manipulate with not a sufficient number of data. The all test cases for neural network training provided sufficient values of convergance error. However, the learning was evaluated based on data from simple system. It would be interesting to do further research with more complex system.

Chapter 7

Resumé

Cieľom tejto práce bolo navrhnúť a implementovať systém, ktorý je schopný vykonávať optimalizačné úlohy a strojové učenie v decentralizovanom prostredí s garanciou privátnosti dát každého agenta. Spomínaný návrh a implementácia bola prevedená na niekoľkých úlohách s uspokojivými výsledkami ako je popísané v experimentálnej časti tejto práce.

Prvým z krokov bolo riešiť optimalizačný problém systému, ktorý sa skladá z viacerých výpočtových jednotiek (agentov) navzájom pospájaných a dokopy vytvárajúcich sieť, kde každý agent disponuje svojími privátnymi citlivými dátami, ktoré sú dôležité pre optimalizáciu bežiacu v spomínanej sieti. Práve preto bol predstavený koncept distribuovanej optimalizácie a moderných algoritmov na jej riešenie. Tie najčastejšie používané, ADMM a ALADIN, boli implementované v softvéri MATLAB na príkladoch s konvexnými ale aj nekonvexnými účelovými funkciami. V oboch prípadoch, ALADIN metóda vyšla ako preferovaná, pretože vykazovala rýchlejšiu mieru konvergencie. Toto tvrdenie je založené na výsledkoch problému fitovania dát a problému naplánovania udalosti.

Ak uvažujeme presnosť riešenia, obidve metódy boli porovnateľne dobré. Oba algoritmy boli ovplyvňované stopovacím kritériom, hlavne vybranou hodnotou tolerancie. Táto hodnota sa menila na základe definície optimalizačného problému. Okrem toho boli výsledky ovplyvnené aj výberom algoritmu pre lokálnu optimalizáciu, ktorého typ

závisí od konvexnosti účelovej funkcie.

Okrem tohoto bolo realizované aj porovnanie centralizovaného a decentralizovaného prístupu pre všetky problémy a každú metódu. Na základe výsledkov, decentralizovaná optimalizácia vykazuje pomalšiu konvergenciu ako centralizovaná. Je to spôsobené nepriamou výmenou dát v sieti. Napriek tomu presnosť riešenia bola takmer rovnaká ako pri centralizovanom spôsobe. Toto považujeme za pozitívnu informáciu, pretože to znamená, že centrálna jednotka nie je potrebná na dohliadanie na optimalizačný proces v sieti. Naopak, môžeme čerpať z výhod decentralizovaného prístupu, akými sú napríklad nižšia výpočtová záťaž a menšie pamäťové miesto výpočtovej jednotky, eliminácia jedného bodu zlyhania, atď.

Pochopenie distribuovanej a decentralizovanej optimalizácie bolo len prvým krokom. Hlavným cieľom bola aplikácia týchto vedomostí na algoritmy strojového učenia. Rozhodli sme sa použiť umelé neurónové siete ako model, ktorý môže byť natrénovaný nie len distribuovaným ale aj decentralizovaným spôsobom. Niečo takéto môže byť použité v mnohých moderných aplikáciach v oblasti zdravotníctva, bankovníctva, a pod. Výsledok našich algoritmov distribuovaného trénovania bola neurónová sieť, ktorá poskytuje veľmi presné výstupné hodnoty, na základe informácie CE parametra. Použili sme viacero špeciálnych príkladov na odtestovanie robustnosti distribuovaného trénovania, ako napríklad nerovnomerné rozdelenie dát alebo dynamickú zmenu v aktívnych agentoch. Oba prípady skonvergovali počas 100 iterácii s chybou konvergencie, ktorá bola považovaná za uspokojivú.

Nakoniec bolo prevedené porovnanie centralizovaného a decentralizovanho učenia. Chyba konvergencie centralizovaného učenia bola menšia v porovnaní s decentralizovaným a to isté platí aj pre počet iterícii. Pomalšia konvergencia decentralizovaného prístupu bola očakávaná, ale také zníženie presnosti riešenia nebolo predpokladané. Každopádne, chyba konvergencia bola počítaná ako rozdiel predikovaného a cieľového výstupu na 350 vzorkách, takže je to stále tolerovateľné.

Ak si to zhrnieme, algoritmus centralizovanej optimalizácie môže byť úplne nahradený decentralizovaným, ak pomalšia konvergencia nie je prekážkou. Výber metódy distribuovanej optimalizácie závisí od definície problému. Aj keď je ALADIN metóda preferovaná v našich testovacích príkladoch, nastavenie H a g, pre úlohy vyznačujúce sa obavami o ochranu súkromia, môže predstavovať výzvu. Toto je niečo, čo by mohlo byť ďalej skúmané za cieľom vyhnutia sa pracnému ladeniu.

Predošlé tvrdenie o nahradení centralizovaného prístupu decentralizovaných platí aj pre strojové učenie. Naviac, pri trénovaní neurónových sietí musíme byť opatrní, čo sa týka počtu agentov v sieti. Podľa Sek. 5.1.1 platí, že čím je väčší počet agentov v sieti, tým je presnosť a miera konvergencie horšia. Ale toto platí iba pre prípady, kde je jeden dataset rozdelený medzi vysoký počet agentov alebo agenti manipulujú s nedostatočným množstvom dát. Všetky príklady trénovania neurónových sietí predkladané v tejto práci, poskytovali uspokojivé hodnoty chyby konvergencie. Napriek tomu, učiaci proces bol vykonávaný na základe dát z jednoduchého systému. Bolo by zaujímavé previesť ďalší prieskum so zložitejším systémom.

Bibliography

[1] T. Addair. Decentralized and distributed machine learning model training with actors. Online. Accessed: 2019-22-05, `http://www.scs.stanford.edu/17au-cs244b/labs/projects/addair.pdf`.

[2] T. Ayodele. *Types of Machine Learning Algorithms*. 02 2010.

[3] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, January 2011.

[4] L. Camargo and T. Yoneyama. Specification of training sets and the number of hidden neurons for multilayer perceptrons. *Neural Computation*, 13:2673–2680, 12 2001.

[5] X. Chen, J. Ji, C. Luo, W. Liao, and P. Li. When machine learning meets blockchain: A decentralized, privacy-preserving and secure design. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1178–1187, 2018.

[6] S. Dreiseitl and L. Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, 35(5):352 – 359, 2002.

[7] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational

problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17 – 40, 1976.

[8] R. Hecht-Nielsen. Theory of the backpropagation neural network based on nonindent by robert hecht-nielsen, which appeared in proceedings of the international joint conference on neural networks 1, 593–611, june 1989. © 1989 IEEE. In *Neural Networks for Perception*, pages 65–93. Elsevier, 1992.

[9] S. Herzog, C. Tetzlaff, and F. Wörgötter. Evolving artificial neural networks with feedback. *Neural Networks*, 123:153 – 162, 2020.

[10] B. Houska, J. Frasch, and M. Diehl. An augmented lagrangian based algorithm for distributed nonconvex optimization. *SIAM Journal on Optimization*, 26, 04 2016.

[11] A. K. Jain, J. Mao, and K. Mohiuddin. Artificial neural networks: A tutorial. *IEEE Computer*, 29:31–44, 1996.

[12] E. Kostarelou and G. K. D. Saharidis. In IGI Global J. Wang, editor, *Encyclopedia of Business Analytics and Optimization*, chapter Centralize vs. Decentralize Supply Chain Analysis, page 429–439. 2014.

[13] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica (Ljubljana)*, 31, 10 2007.

[14] I. Necoara, D. Doan, and J. Suykens. Application of the proximal center decomposition method to distributed model predictive control. volume 290, pages 2900–2905, 01 2008.

[15] M. A. Nielsen. Neural networks and deep learning, 2018.

[16] A. Rantzer. Dynamic dual decomposition for distributed control. In *2009 American Control Conference*, pages 884–888, June 2009.

[17] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.

[18] J. Tsitsiklis. Problems in decentralized decision making and computation. 08 2005.

[19] T. Yang, X. Yi, J. Wu, Y. Yuan, D. Wu, Z. Meng, Y. Hong, H. Wang, Z. Lin, and K. H. Johansson. A survey of distributed optimization. *Annual Reviews in Control*, 47:278 – 305, 2019.

[20] C. Zhang, Q. Li, and P. Zhao. Decentralized optimization with edge sampling. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 658–664. International Joint Conferences on Artificial Intelligence Organization, 7 2019.