P-split formulation for neural networks

Venla Kjäll

School of Science

Bachelor's thesis
Espoo 25.6.2024

Supervisor

Prof. Fabricio Oliveira

Advisor

M.Sc. (Tech.) Liu Yu

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the bachelor's thesis

Author Venla Kjäll

Title P-split formulation for neural networks

Degree programme Bachelor's Programme in Science and Technology

Major Mathematics and Systems Sciences **Code of major** SCI3029

Teacher in charge Prof. Fabricio Oliveira

Advisor M.Sc. (Tech.) Liu Yu

Date 25.6.2024 **Number of pages** 19 **Language** English

Abstract

Deep neural networks (DNNs) have become highly popular machine learning (ML) models. Multiple factors have accelerated their rise in popularity. First of all, DNNs have proven to be powerful tools in various tasks across a variety of fields, including natural language processing, which is currently in high demand. In addition, the structure and functionality of the neural networks are inspired by the human brain, thus making them naturally intriguing for people. This popularity has led to extensive research and development of the DNNs.

The mathematical optimization of trained neural networks is an area of research that has provoked significant interest due to its promising applications. However, as a result of their non-linear nature, DNNs are challenging to optimize. Therefore, different mathematical formulations that exactly model the structure and behavior of DNNs are employed. Beyond the classic use of the 0-1 mixed-integer linear programming (0-1 MILP) formulation, a new and promising class of formulations, known as the P-split formulations, has been proposed. P-split formulations make use of disjunctive programming, which is the combination of linear programming with disjunctive constraints arising from the need to model logical conditions.

This thesis expands the research focusing on P-split formulations by conducting a series of computational experiments to assess the efficiency of P-split formulations in optimizing neural networks with a more general architecture. In the computational experiments, the solution times and linear relaxations of the P-split formulations are compared against those of the classic 0-1 MILP formulation. The results show that the P-split formulations do not consistently offer computational efficiency advantages over the 0-1 MILP formulation. However, they provide tighter relaxations, demonstrating the potential of the formulations. These findings suggest that further research is needed to develop better-performing formulations for a broader range of network structures.

Keywords Deep neural networks, Mathematical optimization, Mixed-integer linear programming, Mixed-integer formulations, Disjunctive programming, Disjunctive constraints

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Tekniikan kandidaatintyön tiivistelmä

Tekijä Venla Kjäll

Työn nimi P-split formulation for neural networks

Koulutusohjelma Teknistieteellinen kandidaattiohjelma

Pääaine Matematiikka ja systeemitieteet **Pääaineen koodi** SCI3029

Vastuuopettaja Prof. Fabricio Oliveira

Työn ohjaaja M.Sc. (Tech.) Liu Yu

Päivämäärä 25.6.2024 **Sivumäärä** 19 **Kieli** Englanti

Tiivistelmä

Syväoppivat neuroverkot ovat saavuttaneet suuren suosion koneoppimismalleista kiinnostuneiden keskuudessa. Tämän suuren suosion kehittymiseen ovat vaikuttaneet lukuisat eri tekijät. Syväoppivat neuroverkot ovat ensinnäkin osoittaneet tehokkuutensa monipuolisissa käyttökohteissa useilla eri aloilla, kuten luonnollisen kielen käsittelyjärjestelmissä, joiden kysyntä on erityisen suurta tällä hetkellä. Lisäksi ihmisen luonnollista mielenkiintoa neuroverkoja kohtaan lisää se, että niiden rakenne ja toiminnallisuus pohjautuvat ihmisaivoihin. Tämä edellä kuvattu suuri suosio on johtanut neuroverkkojen laajamittaiseen tutkimiseen ja kehittämiseen.

Koulutettujen neuroverkkojen matemaattinen optimointi on tieteenala, joka on herättänyt erityistä mielenkiintoa sen lupaavien sovelluskohteiden vuoksi. Neuroverkkojen epälineaarisen rakenteen takia niiden optimointi on kuitenkin haastavaa. Tämän vuoksi optimoinnissa usein käytetään neuroverkkojen sijaan matemaattisia malleja, jotka kuvaavat tarkasti niiden rakennetta ja käyttäytymistä. Klassisen lineaarisen sekalukuoptimointimallin lisäksi on kehitetty uusi ja lupaava kategoria matemaattisia malleja nimeltään P-split-mallit. P-split-mallit hyödyntävät disjunktiivista ohjelmointia (engl. disjunctive programming). Disjunktiivisessa ohjelmoinnissa mallinnetaan rajoitteita, jotka johtuvat tarpeesta mallintaa loogisia ehtoja, kuten "joko-tai".

Tämä kandidaatintyö pyrkii jatkamaan P-split-malleihin kohdistuvaa tutkimustyötä tarkastelemalla niiden tehokkuutta rakenteeltaan yleisluontoisempien neuroverkkojen optimoinnissa. Laskennallisten tutkimusten perusteella selvitettiin, kuinka P-split-mallit vertautuvat klassiseen lineaariseen sekalukuoptimointimalliin tutkittaessa mallien pohjalta luotujen optimointiongelmien ratkaisuaikoja sekä lineaarisia relaksaatioita (engl. linear relaxations). Tuloksista voitiin huomata, että P-split-mallit eivät ole selvästi klassista mallia nopeampia ratkaista. Toisaalta P-split-mallit osoittivat potentiaalinsa tuottamalla tiukempia lineaarisia relaksaatiota. Näiden tutkimustulosten perusteella voidaan siis todeta, että yhä on syytä jatkaa tutkimusta, jossa tarkoituksena on löytää laskennallisesti tehokkaampia ja toimivampia malleja neuroverkkojen optimointiin.

Avainsanat Syväoppiminen, Matemaattinen optimointi, Lineaarinen sekalukuoptimointi, Sekalukuoptimointimalli, Disjunktiivinen ohjelmointi, Disjunktiiviset rajoitteet

Contents

1 Introduction

Deep neural networks (DNNs) have been established as one of the most effective machine learning (ML) models in estimating non-linear relationships between input and output features and recognizing complex patterns in the data. They are used for countless tasks in a variety of fields such as robotics (Nguyen and Cheah, 2022), speech and audio processing (Purwins et al., 2019), and even healthcare (Kollias et al., 2018). A DNN consists of layers of neurons. Generally, there are three types of layers: an input layer, an output layer, and hidden layers in between. Each neuron in the hidden layers applies a linear transformation to the outputs of the neurons in the preceding layer and then passes the result through a non-linear activation function. The resulting value is called an activation (or output) of the neuron that is propagated to the following layer.

Due to their popularity, the properties of DNNs are under extensive research. However, as a result of their non-linear nature, DNNs are particularly challenging to optimize. Thus, an intriguing research topic is the use of mixed-integer linear programming (MILP) formulations as exact mathematical representations for DNNs. These mathematical models can be used for analyzing trained neural networks, for instance, by computing optimal input values according to a certain objective function or building adversarial examples to identify hidden weaknesses of the DNN in question (Fischetti and Jo, 2018).

One specific area of interest is DNNs that use the rectified linear unit (ReLU) as the activation function. ReLU is a simple yet powerful activation function that calculates the maximum between zero and its input (Nair and Hinton, 2010). The behavior of a ReLU function in a neural network can be programmed using different strategies. The classic method is to introduce one binary variable for each neuron that uses the ReLU activation and apply the big-M method. This classic formulation is called the 0-1 MILP formulation (Fischetti and Jo, 2018; Grimstad and Andersson, 2019). The more recent approach is to consider the ReLU function as a disjunction and use the partition-based P-split formulations, which have exhibited promising results by balancing the model size and tightness in optimization problems related to image classification (Tsay et al., 2021; Kronqvist et al., 2022).

In this study, we consider the P-split formulations for ReLU-NNs with a more general architecture. We compare the computational efficiency of P-split formulations with different parametrizations to the classic 0-1 MILP formulation. The solution times as well as the strength of the linear relaxation of each problem are analyzed.

This thesis is structured as follows. Section 2 reviews the previous research on mathematical modeling and optimization of DNNs. Section 3 provides an overview of the methodology introducing the 0-1 MILP and P-split formulations. Section 4 details the design of the computational experiments and presents the results of the study. Section 5 concludes the thesis by summarizing the key results, discussing the limitations, and suggesting directions for future research.

2 Literature review

Numerous applications for solving an optimization problem containing trained neural networks have been introduced in the literature. One important application is verifying the robustness of NNs used in safety-critical systems such as self-driving cars or recognition of voice commands. In Belotti et al. (2016) three attack algorithms and adversarial examples for NNs were introduced. The 100% effectiveness of these algorithms illustrates the need for better techniques to evaluate the robustness of neural networks. Other interesting applications include optimizing an unknown function approximated by a trained NN and generating new images optimizing over neural networks trained for visual perception tasks (Gatys et al., 2015). However, this need for optimizing over complex trained neural networks does not come without difficulties.

Today, many challenges in the optimization of DNNs relate to the indicator constraints arising from the piecewise linear activation functions. In addition to the ReLU function and its variations, other piecewise linear activation functions include hard tanh and maxpooling. In Huchette et al. (2023), the authors analyzed the challenges of optimizing over trained ReLU-NNs using polyhedral theory and the associated optimization methodologies such as MILP. This literature indicates that once the parameters of a network have been fixed, the NN is simply a piecewise linear function if the activation function of each node is piecewise linear, making optimization over these NNs a piecewise linear optimization problem. Given this inherent linearity, MILP becomes a suitable tool in tackling these optimization problems.

MILP provides an effective framework for finding provably optimal solutions to nonconvex piecewise linear functions. The paper further discusses exact models for these functions using MILP. The authors mention that while the Operations Research community has extensively developed piecewise linear optimization methods based on MILP, many of these methods are for separable linear functions with only few dimensions, thus inapplicable to modeling neurons in a neural network. A common technique to reformulate disjunctive constraints is the big-M method, used in the 0-1 MILP formulation for neural networks. However, this method often results in weak convex relaxations that can slow down the convergence of the solution and lead to intractable problems.

The other extreme is the extended convex hull formulation for disjunctions, as detailed in Balas (1998). This approach can achieve the strongest possible convex relaxation for a single unit, but results in a larger formulation. In addition, this formulation usually performs worse than anticipated. Anderson et al. (2020) developed an ideal non-extended formulation that was compared against the extended formulation. Computational experiments showed that for a small ReLU-NN, the ideal extended formulation was five times slower than the non-extended. In the case of a larger network, Gurobi Optimizer failed to solve the extended method within the 1800s time limit due to the extensive growth of the formulation.

These extended formulations addressing the modeling of indicator constraints using disjunctive programming were also pioneered by Balas (1985, 1998). Disjunctive

programming is the combination of linear programming with disjunctive constraints and provides concepts for obtaining convex approximations of discrete optimization problems. Balas (1985) introduced a general framework for classifying and ranking linear programming relaxations by computational cost. The class of relaxations establishes a hierarchy that ranges from the usual linear programming relaxation to the convex hull of the feasible set itself.

Inspired by disjunctive programming and hierarchical relaxations, Kronqvist et al. (2021, 2022) developed a class of formulations in between the big-M method and the extended convex hull for general disjunctive programs, termed P-split formulations. Tsay et al. (2021) in turn developed a similar hierarchical class of formulations specifically for trained ReLU-NNs. The key idea of the formulations is to partition the variables into groups and use disjunctive programming to form the convex hull over the partitions. In this thesis, we further explore the P-split formulations for neural networks, aiming to uncover their potential and limitations in enhancing the optimization of ReLU-NNs..

In addition to the MILP-based methods, further options for encoding indicator constraints have been introduced in the context of other supervised machine learning methods. For example, Belotti et al. (2016) introduced a nonlinear and nonconvex mixed integer non-linear programming (MINLP) reformulation for Support Vector Machine (SVM) methods with the ramp loss function. The empirical results showed that a class of classification problems can be solved more efficiently using the MINLP formulation. Also, the study emphasized the importance of aggressive bound tightening (BT), a crucial tool in MINLP, in tackling the known challenges of MIP formulations as well. The significance of bound tightening has already been addressed in studies concerning the optimization of trained DNNs.

Grimstad and Andersson (2019) presented several BT procedures for the 0-1 MILP formulation by Fischetti and Jo (2018). The procedures are divided into two categories: feasibility-based bound tightening (FBBT) and optimization-based bound tightening (OBBT). A computational study is provided, focusing on the solution times of three different optimization problems involving the 0-1 MILP formulation and BT procedures. The BT procedures can be computationally expensive, thus the authors suggest pre-computing the bounds and saving them for potential reuse. However, the results show that bound tightening procedures can significantly improve the solution times, even though the best-performing BT procedure depends on the problem.

Furthermore, in the partition-based formulation for ReLU-NNs by Tsay et al. (2021) bounds on the partitions play an important role in the tightness of the model. In the computational experiments, OBBT was implemented by tightening bounds for all auxiliary variables. The results showed that the BT procedure improved all models since they consider interdependencies among all inputs within a partition. If BT procedures are not under consideration, the variable bounds for P-split formulations can also be computed using interval arithmetic that gives valid but not tightest possible bounds.

3 Methodology

This section covers the 0-1 MILP formulation, introduction to disjunctive programming, the general form of the P-split formulation, and the special case of the formulation for the ReLU neural networks. For simplicity, we introduce only the formulation for disjunctions with one constraint per disjunct, but the formulation can easily be extended to multiple constraints per disjunct (Kronqvist et al., 2022). The performance of the 0-1 MILP and P-split formulations for neural networks are compared in the computational experiments in the following section.

3.1 The 0-1 MILP formulation

This classic formulation was implemented following the paper by (Grimstad and Andersson, 2019). We consider ReLU DNNs consisting of $K+1$ layers that are labeled with numbers from 0 to K. The layer 0 corresponds to the input layer of the neural network, which is not typically calculated as a separate layer. Whereas the layer K denotes the output layer of the network. Each layer $k \in \{0, ..., K\}$ comprises n_k neurons numbered from 1 to n_k.

Let $x^k \in \mathbb{R}^{n_k}$ denote the output vector of layer k. Since layer 0 is the input layer of the DNN, x^0 corresponds to the input vector of the network, where x_j^0 is the j-th input value. Respectively, x^K is the output vector of the DNN and x_j^K is the j-th output value. For each hidden layer in the ReLU DNN, the output of the neuron is calculated according to the formula

$$x^k = \mathrm{ReLU}(W^{k-1}x^{k-1} + b^k), \tag{1}$$

where $\mathrm{ReLU}(y) = \max(0, y)$ for each vector y componentwise. We should note that because of the ReLU function, all output vectors x^k are positive even though the weight and bias matrices may contain negative entries. The exceptions are the vector x^0, which represents the input of the network as a whole, and the output layer of the network which uses the identity activation and is thus calculated as

$$x^K = W^{K-1}x^{K-1} + b^K, \tag{2}$$

without the ReLU function.

To represent the ReLU function as a set of 0-1 MILP constraints, we examine the linear equation

$$w^T x + b = x - s, \qquad x, s \geq 0, \tag{3}$$

where x represents the positive part and s the negative part of the output of the ReLU function. Assuming that we can obtain finite bounds L and U such that $L \leq w^T x + b \leq U$, the ReLU logic can be imposed using the big-M constraints

$$\begin{aligned} x &\leq Uz \\ s &\leq -L(1-z) \\ z &\in \{0, 1\} \end{aligned}$$

where z is a binary activation variable used to encode the activation behavior of the ReLU function.

Finally, using the binary activation variable z for each neuron (j,k) and including the big-M constraints we obtain the 0-1 MILP formulation

$$L^0 \leq x^0 \leq U^0 \tag{5a}$$

$$\left.\begin{array}{l} W^{k-1}x^{k-1} + b^k = x^k - s^k \\ x^k, s^k \geq 0 \end{array}\right\} \; K = 1, ..., K-1 \tag{5b}$$

$$z_j^k \in \{0,1\} \qquad k = 1, ..., K-1 \quad j = 1, ..., n_k \tag{5c}$$

$$\left.\begin{array}{l} x_j^k \leq U_j^k z_j^k \\ s_j^k \leq -L_j^k(1 - z_j^k) \end{array}\right\} \; K = 1, ..., K-1, \quad j = 1, ..., n_k \tag{5d}$$

$$W^K x^{K-1} + b^K = x^K \tag{5e}$$

$$L^K \leq x^K \leq U^K. \tag{5f}$$

The set of constraints (5) represents a trained ReLU DNN. The constraint (5a) bounds the input variables and the constraint (5e) represents the output of the network. Consequently, constraints (5b-5d) represent the activations of the hidden layers.

3.2 Disjunctive programming and the P-split formulation

Disjunctive programming refers to mathematical programming with disjunctive constraints that arise from logical conditions, such as conjunction ("and"), disjunction ("or"), and negation ("complement of"). First, we present the general form of the P-split formulation that is applicable to MILP problems that contain constraints with a clear disjunctive structure

$$\bigvee_{l \in D} [g_k(\boldsymbol{x}) \leq b_{l,k} \quad \forall k \in C_l], \qquad \boldsymbol{x} \in \chi \subset \mathbb{R}^n, \tag{6}$$

where χ is a convex compact set, D includes the disjunct indices, and C_l contains the constraint indices of disjunct l. To derive the P-split formulations following Kronqvist et al. (2022), we make three additional assumptions about the structure of the problem.

1. The functions $g_k : \mathbb{R}^n \to \mathbb{R}$ are convex and additively separable, i.e., $g_k(\boldsymbol{x}) = \sum_{i=1}^n h_{ik}(x_i)$ where $h_{ik} : \mathbb{R} \to \mathbb{R}$ are convex functions and each disjuction is non-empty over χ.

2. The functions g_k are bounded over χ.

3. Each disjunct contains significantly fewer constraints than variables in each disjunction, that is, $|C_l| << n, \forall l \in D$.

The third assumption specifies the problem structures that are advantageous for the P-split formulations.

The key idea of the P-split formulation is to split the variables of each disjunction into P sets. All variables must be included in precisely one partitioning and index set $I_1, ..., I_P$, that is, $I_1 \cup ... \cup I_P = 1, ..., n$ and $I_i \cap I_j = \emptyset,\ i \neq j$. This partitioning operation lifts the disjunction into a higher dimension in a disaggregated form while effectively linearizing the constraints. Based on the variable partitioning, $P \times |D|$ auxiliary variables $\alpha_s^l \in \mathbb{R}$ are introduced together with the constraints

$$\sum_{i \in I_s} h_{i,l}(x_i) \leq \alpha_s^l \qquad \forall s \in \{1, ..., P\},\ \forall l \in D. \tag{7}$$

With the constraints (7) and the auxiliary variables α_s^l the disjunction (6) can be presented in a disaggregated form

$$\bigvee_{l \in D} [g_k(\boldsymbol{x}) \leq b_{l,k}] \longrightarrow \bigvee_{l \in D} \begin{bmatrix} \sum\limits_{i \in I_1} h_{i,l}(x_i) \leq \alpha_1^l \\ ... \\ \sum\limits_{i \in I_P} h_{i,l}(x_i) \leq \alpha_P^l \\ \sum\limits_{s=1}^{P} \alpha_s^l \leq b_l \end{bmatrix}, \qquad \boldsymbol{x} \in \chi, \boldsymbol{\alpha}^l \in \mathbb{R}^P\ \forall l \in D. \tag{8}$$

The reformulation (8) splits (or disaggregates) the constraint in the disjunction (6) into P parts.

Next, the disjunction (8) is relaxed by considering the split constraints as global constraints.

$$\begin{aligned} &\bigvee_{l \in D} \begin{bmatrix} \sum\limits_{s=1}^{P} \alpha_s^l \leq b_l \\ \underline{\alpha}_s^l \leq \alpha_s^l \leq \overline{\alpha}_s^l \quad \forall s \in \{1, ..., P\} \end{bmatrix} \\ &\sum_{i \in I_s} h_{i,l}(x_i) \leq \alpha_s^l \qquad \forall s \in \{1, ..., P\},\ \forall l \in D \\ &\boldsymbol{x} \in \chi, \boldsymbol{\alpha}^l \in \mathbb{R}^P \qquad \forall l \in D. \end{aligned} \tag{9}$$

The formulation (9) is a P-split representation of the original disjunction (7), where the auxiliary variable bounds $\underline{\alpha}_s^l,\ \overline{\alpha}_s^l$ are defined as follows

$$\underline{\alpha}_s^l := \min_{\boldsymbol{x} \in \chi} \sum_{i \in I_s} h_{i,l}(x_i), \qquad \overline{\alpha}_s^l := \max_{\boldsymbol{x} \in \chi} \sum_{i \in I_s} h_{i,l}(x_i). \tag{10}$$

Tight bounds are not required for the P-split formulations, but weaker bounds result in weaker relaxations. Lastly, taking the extended convex hull (Balas, 1998) of the

disjunction (9) produces the P-split formulation

$$\alpha_s^l = \sum_{d \in D} \nu_d^{\alpha_s^l} \qquad \forall s \in 1, ..., P,\ \forall l \in D \tag{11a}$$

$$\sum_{s=1}^{P} \nu_l^{\alpha_s^l} \leq b_l \lambda_l \qquad \forall l \in D \tag{11b}$$

$$\underline{\alpha}_s^l \lambda_d \leq \nu_d^{\alpha_s^l} \leq \overline{\alpha}_s^l \lambda_d \qquad \forall s \in 1, ..., P,\ \forall l, d \in D \tag{11c}$$

$$\sum_{i \in I_s} h_{i,l}(x_i) \leq \alpha_s^l \qquad \forall s \in 1, ..., P,\ \forall l \in D \tag{11d}$$

$$\sum_{l \in D} \lambda_l = 1, \qquad \boldsymbol{\lambda} \in 0, 1^{|D|} \tag{11e}$$

$$\boldsymbol{x} \in \chi, \boldsymbol{\alpha}^l \in \mathbb{R}^P, \boldsymbol{\nu}^{\alpha_s^l} \in \mathbb{R}^{|D|} \qquad \forall s \in 1, ..., P,\ \forall l \in D. \tag{11f}$$

3.3 The P-split formulation for ReLU-NNs

The behavior of each ReLU node can be represented as a disjunction

$$\begin{bmatrix} y = \boldsymbol{w}^T \boldsymbol{x} + b \\ \boldsymbol{w}^T \boldsymbol{x} + b \geq 0 \end{bmatrix} \vee \begin{bmatrix} y = 0 \\ \boldsymbol{w}^T \boldsymbol{x} + b \leq 0 \end{bmatrix} \tag{12}$$

where $\boldsymbol{w}$ contains the weights, $\boldsymbol{x}$ is the input vector and b is the bias. The variable y determines the output of the node. This disjunction can be modeled as the P-split formulation with a few simplifications. First, we denote

$$\sum_{i \in I_s} h_i(x_i) = \sum_{i \in I_s} w_i x_i \tag{13}$$

and since here $\sum_{i \in I_s} h_i$ is affine, then (11d) can be reinforced to an equality constraint while retaining convexity. Secondly, we can introduce only one shared α variable for both of the disjuncts since both of the disjuncts contain the same sum of functions (13) up to a scaling factor. Then we can use the equality constraint for the two disaggregated variables (11a) to project out the variable $\nu_1^{\alpha_s}$.

The constraint for the output y of the ReLU node is formulated following Tsay et al. (2021). We replace $\boldsymbol{w}^T \boldsymbol{x} = \sum_{s=1}^{P} \nu_2^{\alpha_s}$ in the disjunction (12) and accompany the bias b with the binary variable λ_2 resulting in the P-split formulation for neural networks

$$\alpha_s = \sum_{i \in I_p} w_i x_i \qquad \forall s \in 1, ..., P \tag{14a}$$

$$\sum_{s=1}^{P} (\alpha_s - \nu_2^{\alpha_s}) \leq -b\lambda_1 \tag{14b}$$

$$\sum_{s=1}^{P} \nu_2^{\alpha_s} \geq -b\lambda_2 \tag{14c}$$

$$\underline{\alpha}_s \lambda_1 \leq \alpha_s - \nu_2^{\alpha_s} \leq \overline{\alpha}_s \lambda_1 \qquad \forall s \in 1, ..., P, \ \forall d \in D \tag{14d}$$

$$\underline{\alpha}_s \lambda_2 \leq \nu_2^{\alpha_s} \leq \overline{\alpha}_s \lambda_2 \qquad \forall s \in 1, ..., P, \ \forall d \in D \tag{14e}$$

$$y = \sum_{s=1}^{P} \nu_2^{\alpha_s} + \lambda_2 b \tag{14f}$$

$$\lambda_1 + \lambda_2 = 1. \tag{14g}$$

Constraints (14) represent the behavior of a single ReLU neuron. Subsequently, similarly to the classic 0-1 MILP formulation, we propose these P-split constraints for each neuron (j, k) of the hidden layers in a neural network. Furthermore, the constraints (5a), (5e) and (5f) regarding the input and output of the whole network remain necessary.

4 Computational experiments

In this section, we conducted a series of computational experiments. The experiments aim to investigate how the computational efficiency of the P-split formulation (11) for ReLU-NNs compares to the classic 0-1 MILP formulation (5). In addition to the solution times of each formulation, the root relaxation objective values were recorded to obtain information on the tightness of the formulations. In these experiments, we optimized ReLU-NNs with a more general structure. Previous comparisons have been made solving optimal adversary problems with NNs trained for image classification tasks (Tsay et al., 2021; Kronqvist et al., 2022) where the networks consist of a large input layer and a few small hidden layers.

4.1 Design of the experiments

For the computational experiments, we used the programming language Julia (version 1.10.2) (Julia, 2024) for implementing the P-split formulation (14) as well as building and training the ReLU-NNs. Two ReLU-NNs, medium and large, were built using the Julia package Flux.jl (Flux.jl, 2024). The details of the networks are presented in the Table 1. The networks were trained using the Adam optimizer with a learning rate of 0.0005, and L_2 regularization was applied to prevent overfitting. The concrete compressive strength dataset (Yeh, 2007) was used for training. It contains eight input variables corresponding to different concrete ingredients. The output variable is the concrete hardness in megapascals.

size	layers	number of parameters	MAPE
medium	(8, 64, 32, 1)	2 689	11.94%
large	(8, 128, 64, 32, 1)	11 521	11.11%

Table 1: The trained ReLU-NNs used in the experiments. MAPE refers to the Mean Average Percentage Error of the test set.

The trained neural networks were then formulated into optimization problems where the objective was to find optimal inputs that maximize the concrete hardness. The maximization problem is first formulated using the 0-1 MILP formulation with and without bound tightening. The 0-1 MILP formulation had already been implemented in the Julia package called Gogeta.jl (Gogeta.jl, 2024). Secondly, the problem is formulated as the P-split formulation with a varying number of splits. We used values $P = 2, 3, 4, 6, 8$. Notice that $P = 8$ corresponds to the full split in the first hidden layer. The variable bounds $\underline{\alpha}_s$ and $\overline{\alpha}_s$ are derived using the interval arithmetic. In partitioning the variables, we use a strategy based on node weights (Tsay et al., 2021). First, the weights of the variables are sorted to have the weights in each partition as close as possible. This sorting returns the variable indices, which are then partitioned into groups of equal size (the group size can differ by one if the division did not result in equal parts), i.e., $I_1 = I_2 = ... = I_P$.

The resulting maximization problems were solved using the Gurobi-Optimizer (version 11.0.0) (Gurobi-Optimizer, 2024). The following Gurobi parameter settings **MIPFocus=3** and **Cuts=1** were used to reduce variation in the results and to stay consistent with the computational results in Kronqvist et al. (2022). As previously noted, the focus of the experiments was on the solution times and root relaxation objective values.

The root relaxation is the initial linear program (LP) relaxation at the root node of the branch-and-bound tree, meaning it is the first LP relaxation that Gurobi solves after presolving. The root relaxation objective value depends on the formulation of the optimization problem. It is desirable to have a root relaxation value that is close to the true optimal value since the root relaxation sets the initial bound on the optimal solution, i.e., for a maximization problem the smaller the root relaxation value, the better. Thus, the root relaxation objective value informs us of the tightness of the formulation. The tightness of the relaxation is meaningful because tighter relaxations reduce the search space that is explored by branching.

4.2 Results

Figures 1 and 2 show the resulting solution times for the medium and large NN with different formulations. The solution times of the big-M problems, both with and without bound tightening (BT), are represented as horizontal lines. The big-M outperforms the P-split formulations for both sizes of the neural networks. However, we can notice that the solution times do not grow monotonically because values $P = 4$ and $P = 6$ give faster solution times. This observation aligns with the results in (Tsay et al., 2021), where the intermediate values of N (or P) effectively

balance the model size and tightness for more complex problems. The authors also offered an explanation for this phenomenon. The intermediate values of P can be computationally advantageous because they offer tighter continuous relaxations, but at the same time, the subproblems remain small.

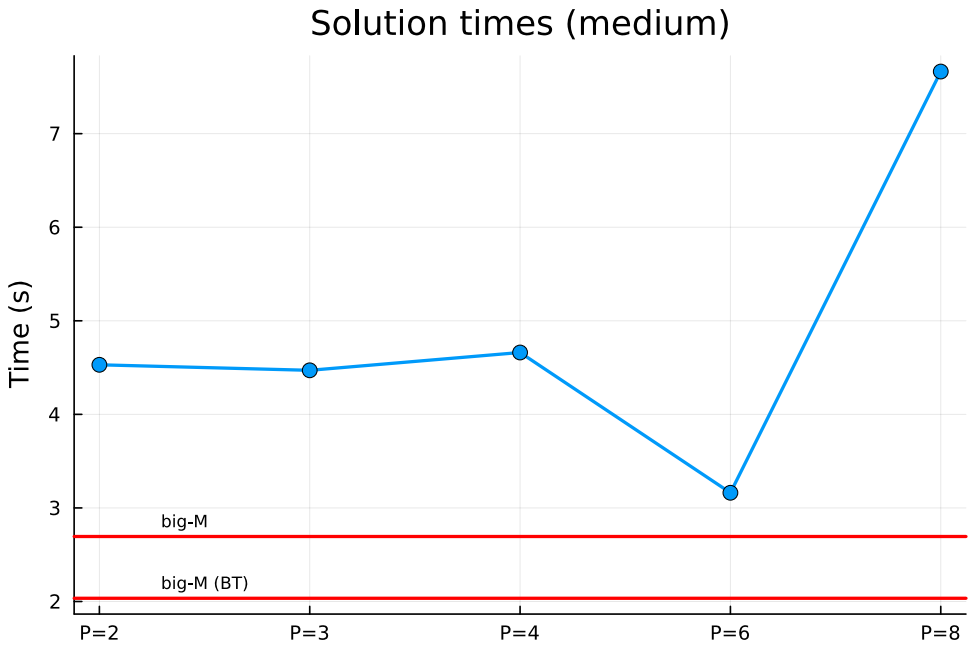

Figure 1: The solution times for the medium ReLU-NN.

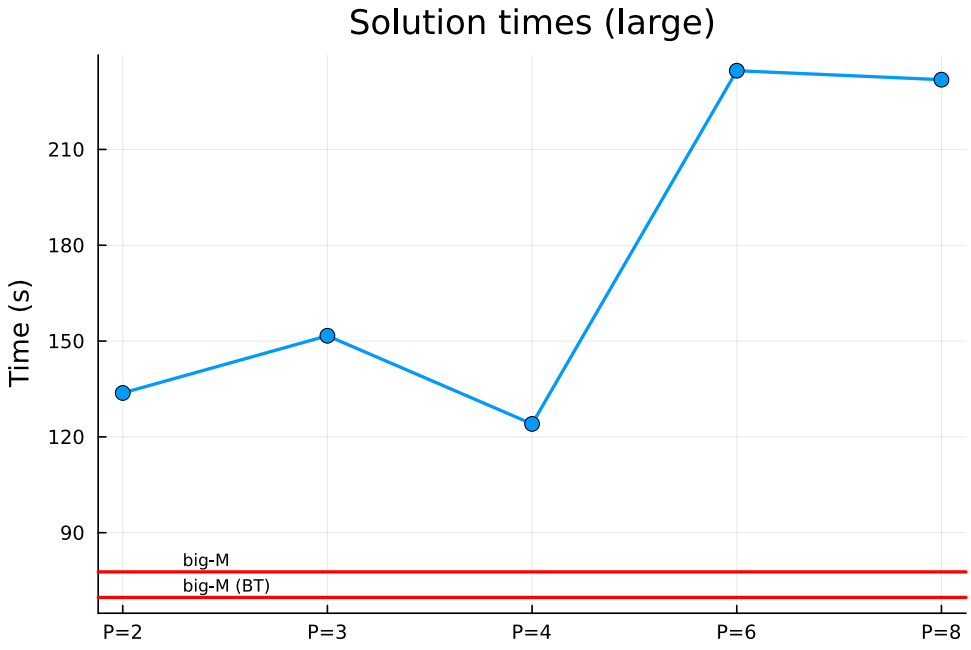

Figure 2: The solution times for the large ReLU-NN.

When we increase the number of splits we end as the convex-hull formulation that provides the strongest possible relaxation. At the same time, the problem size increases as shown in Figure 3, thereby increasing the computational costs. This explains why, when $P = 8$, solving the problem took the longest time.

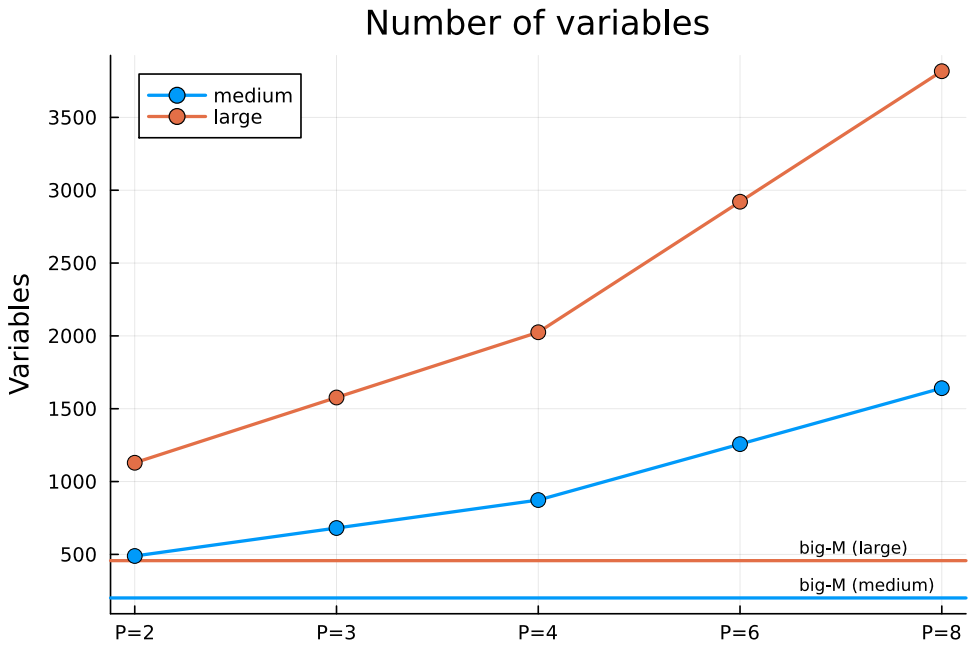

Figure 3: Number of variables in the different formulations.

Figure 4 presents the root relaxation objective values of the formulations. From the figure, it can be seen that the P-split formulations yield lower values than the big-M formulation. Since we are solving a maximization problem, lower values are preferred and indicate tighter formulations. This result is expected because the P-split formulations are designed to fall in between the big-M and convex-hull when considering the model tightness. The tighter the model, the faster it tends to converge to the optimal solution. However, in this case, the tightness is achieved by introducing additional variables and constraints. Although the model is tighter, handling more variables and constraints diminishes the computational benefits.

These results show the importance of the third assumption that characterizes the problem structures that are advantageous for P-split formulations. In the case of ReLU-NNs, the architecture of the network defines the problem structure. If the NN has a large input layer, it leads to a situation where we have disjunctive constraints involving a large number of variables. The number of variables and constraints that are introduced by the P-split formulations depends on the number of partitions and disjunctive terms instead of the number of variables within each disjunctive constraint. If the number of variables is large, using a P-split formulation is more beneficial. However, in a more general case, as demonstrated in these computational experiments, it is not as effective compared to the classic formulation. Although the P-split formulation implemented in this thesis might not be the most compact version, it still demonstrates the phenomenon.

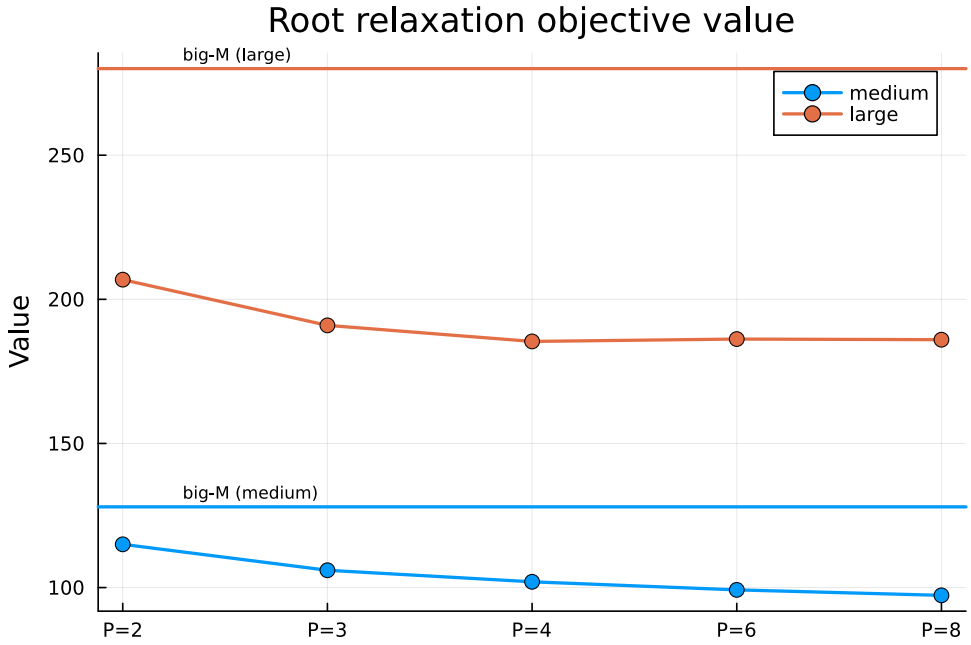

Figure 4: The root relaxation objective values of the different formulations.

5 Conclusion

In this thesis, the P-split formulation (14) for ReLU-NNs was implemented following Kronqvist et al. (2022) and Tsay et al. (2021), and its computational efficiency was assessed with more general network architectures. The P-split formulation had previously shown rather impressive results in optimizing trained ReLU-NNs, as well as other problems, including K-means clustering and P-ball problems. The performance of the classic 0-1 MILP formulation (5) by (Grimstad and Andersson, 2019) served as the baseline against which the performance of the P-split formulation was compared. The comparison was conducted through a series of computational tests, focusing on solution times and root relaxation objective values of each formulation.

The results of the computational experiments showed that although P-split formulation performs well in optimizing some network architectures (large input size with a few layers), its overall performance is not as strong. In fact, its performance appears to be inferior to the classic MILP formulation. On the other hand, the results demonstrated that for some values of P, the formulation was faster to solve compared to other values. This observation indicates that partitioning the variables can indeed help balance the model size and tightness well, improving its tractability. The P-split formulations also provided stronger relaxations than the classic formulation.

The performance of the P-split formulation could be further improved by employing different bound tightening procedures or partitioning strategies as discussed in Tsay et al. (2021). In the scope of this thesis, we considered only one partitioning strategy based on node weights that is relatively easy to implement and outperforms the random partitioning strategy. BT procedures were also beyond the scope of this thesis. Hence, the auxiliary variable bounds were calculated using simple interval

arithmetic. However, bound tightening of auxiliary variable bounds could be a valuable tool in achieving better-performing formulations, as argued in the same paper by Tsay et al. (2021).

Other limitations include only training two ReLU-NNs with similar architectures, differing only in size. Also, only one dataset, the concrete compressive strength (Yeh, 2007), was used in training the networks. Additional observations on the performance of the P-split formulations could have arisen by experimenting with the number of hidden layers and nodes. The mean average performance errors of the trained networks were quite high as shown in the table 1. However, the errors should not affect the computational results since we were not interested in the accuracy of the optimal solutions in this thesis.

In light of this thesis, it can be concluded that there is much potential in the partition-based P-split formulations. They offer a useful alternative for optimizing ReLU-NNs with a favourable structure. However, due to the popularity and wide use of neural networks, it is important to be able to assess the properties of them more efficiently. Mathematical modeling and optimizing function as important tools in that even though optimizing larger NNs remains a challenge. Consequently, the research for developing better-performing formulations for neural networks and improving the existing ones is still relevant in the future.

References

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1):3–39, 2020.

Egon Balas. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic Discrete Methods*, 6(3): 466–486, 1985.

Egon Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3–44, 1998.

Pietro Belotti, Pierre Bonami, Matteo Fischetti, Andrea Lodi, Michele Monaci, Amaya Nogales-Gómez, and Domenico Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65:545–566, 2016.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

Flux.jl, 2024. URL https://fluxml.ai/Flux.jl/stable/.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

Gogeta.jl, 2024. URL https://github.com/gamma-opt/Gogeta.jl.

Bjarne Grimstad and Henrik Andersson. Relu networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, 2019.

Gurobi-Optimizer, 2024. URL https://www.gurobi.com/.2023.

Joey Huchette, Gonzalo Muñoz, Thiago Serra, and Calvin Tsay. When deep learning meets polyhedral theory: A survey. *arXiv preprint arXiv:2305.00241*, 2023.

Julia, 2024. URL https://julialang.org/.

Dimitrios Kollias, Athanasios Tagaris, Andreas Stafylopatis, Stefanos Kollias, and Georgios Tagaris. Deep neural architectures for prediction in healthcare. *Complex & Intelligent Systems*, 4:119–131, 2018.

Jan Kronqvist, Ruth Misener, and Calvin Tsay. Between steps: Intermediate relaxations between big-m and convex hull formulations. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 299–314. Springer, 2021.

Jan Kronqvist, Ruth Misener, and Calvin Tsay. P-split formulations: A class of intermediate formulations between big-m and convex hull for disjunctive constraints. *arXiv preprint arXiv:2202.05198*, 2022.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Huu-Thiet Nguyen and Chien Chern Cheah. Analytic deep neural network-based robot control. *IEEE/ASME Transactions on Mechatronics*, 27(4):2176–2184, 2022.

Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained relu neural networks. *Advances in neural information processing systems*, 34:3068–3080, 2021.

I-Cheng Yeh. Concrete Compressive Strength. UCI Machine Learning Repository, 2007. DOI: https://doi.org/10.24432/C5PK67.