Journal of Optimization Theory and Applications (2023) 199:931–954
https://doi.org/10.1007/s10957-023-02317-x

Optimization of Sparsity-Constrained Neural Networks as a Mixed Integer Linear Program

NN2MILP

Bodo Rosenhahn[1]

Received: 3 June 2022 / Accepted: 24 September 2023 / Published online: 25 October 2023

Abstract
The literature has shown how to optimize and analyze the parameters of different types of neural networks using mixed integer linear programs (MILP). Building on these developments, this work presents an approach to do so for a McCulloch/Pitts and Rosenblatt neurons. As the original formulation involves a step-function, it is not differentiable, but it is possible to optimize the parameters of neurons, and their concatenation as a shallow neural network, by using a mixed integer linear program. The main contribution of this paper is to additionally enforce sparsity constraints on the weights and activations as well as on the amount of used neurons. Several experiments demonstrate that such constraints effectively prevent overfitting in neural networks, and ensure resource optimized models.

Keywords Mixed integer linear programming · Neural networks · Feature selection · Sparse networks · Resource optimization

1 Introduction

Neural networks are commonly used in AI applications ranging from computer vision, autonomous driving, robotics, games, intelligent production, systems biology, human-computer-interaction to even arts or music. Nearly all existing methods are commonly trained using gradient descent-based optimization and auto-differentiation, so that such optimized systems can be summarized with the term *differentiable programming*, see Baydin et al. [8]. It is well known that the gradient descent-based optimization is prone

Communicated by Bernard Fortz.

✉ Bodo Rosenhahn
rosenhahn@tnt.uni-hannover.de

[1] Leibniz University, Appelstr. 9A, 30165 Hannover, Germany

to convergence problems, overfitting or getting stuck in local minima, so that many different (differential) approaches exist to overcome these issues, e.g., using special optimizers (adam, sgdm, etc.), drop-out layers, etc.

For these reasons, alternative approaches, which allow a global optimization, gained increased attention in the past, as outlined in Section 2. This work follows this research strand by modeling and optimization of a shallow neural network, which consists of non-differentiable neurons. The perceptron model, as it has been originally formulated by Rosenblatt in 1958, is used. As this formulation is not differentiable, this work explains how this model can be mapped to a mixed integer linear program (MILP) for given training data in a supervised learning setting. The network weights can then be (globally) optimized using the simplex algorithm and branch-and-bound or branch-and-cut. The drawback of the proposed method is that the formulation is reasonable memory intensive. It requires many slack variables since all training data and computations through the network are expressed as equality and inequality constraints. Thus, in the current stage the proposed method is mainly suited for small sized datasets.

The optimized network weights can then be assembled to a neural network and used for forward inference on unseen data. The proposed formulation also allows the neat integration of additional constraints on the network to enforce sparsity, efficiency and compactness. In the experiments, examples on enforcing binary or sparseness constraints are presented, which can be quite problematic using traditional differentiable programming. Additionally, a *kill-switch* on neurons to minimize the amount of required neurons during training is presented and finally, the amount of neural activations can also be minimized. Critical parameters in differentiable programming such as random seeds, see Picard [58], dropouts, learning rates, etc. do not exist in the proposed optimization framework. On different datasets competitive performance of the globally optimized sparse MILP neural networks is shown, even though the networks are rather simple and shallow.

To summarize, this work presents the following **contributions**:

1. In line with recent literature, the formulation and optimization of a non-differentiable Rosenblatt perceptron and a corresponding shallow neural network as a MILP for given training data, is presented.
2. This work presents modifications to prior art in MILP models of neural networks by incorporating sparsity constraints on the weights, activations and used neurons. This is useful for explainable AI, the challenge of feature selection and resource optimization.
3. Empirical experiments on a number of datasets, in particular including a study on feature selection, are presented.
4. Software examples for MILP-based optimization of neural networks are provided.[1]

[1] http://www.tnt.uni-hannover.de/staff/rosenhahn/KSDemo.zip.

2 Foundations and Literature

2.1 The Perceptron

In 1943, McCulloch and Pitts formulated their idea for logical calculus using concepts from nervous activities, see McCulloch and Pitts [49]. A McCulloch-Pitts cell with n exciting input lines on which the signals $(x_1 \dots x_n)$ are applied, and m inhibiting input lines on which the signals $(y_1 \dots y_m)$ are applied, calculation works as follows: If $m \geq 1$ and if one of the signals $(y_1 \dots y_m)$ equals 1, the neuron outputs a 0. Otherwise (which is the standard case) the input signals $(x_1 \dots x_n)$ are added to up to $x = \sum x_i$. For $n = 0$, $x = 0$ is set. The value x is compared to the threshold θ. If the values x is greater than or equal to θ, the neuron returns 1, otherwise it returns 0. In 1958, Frank Rosenblatt published his perceptron model which extends the summation to a scalar product, followed by a step function, see Rosenblatt [60]. The perceptron can be summarized as

$$\sigma_j = \begin{cases} 1 : \sum_i \omega_{ij} x_i + b > 0 \\ 0 : \text{else} \end{cases}. \tag{1}$$

The bias value b corresponds to the decision threshold and ω_{ij} are learnable parameters. A combination of such perceptrons in a directed acyclic graph leads to a classic (e.g., fully connected) neural network.

2.2 Mixed Integer Linear Programming

Linear programming (LP) is a method for the minimization (or maximization) of a linear objective function, subject to linear equality or inequality constraints, see Dantzig [17]. Any LP can be expressed in a canonical form as

$$\min\ c^T x \quad \text{s.t. } Ax \leq b, \quad x \geq 0. \tag{2}$$

A standard approach for solving such a LP is the simplex algorithm. Note, that standard solvers also accept equality constraints of the form $A_{\text{eq}} x = b_{\text{eq}}$ which are directly transformed in two inequality constraints of the form $A_{\text{eq}} x \leq b_{\text{eq}}$ and $-A_{\text{eq}} x \leq -b_{\text{eq}}$. Thus, the constraint matrix A refers to the inequality constraints and A_{eq} to the equality constraints for better readability. It is also possible to allow for the optimization of integer constraints, see Murty [52] or Dennis et al. [18], which is then called a mixed integer linear program (MILP). The solvers then use concepts such as branch-and-cut or branch-and-bound.

A (MI)LP is a very powerful tool, which allows the efficient optimization of graph problems, e.g., graph cut, graph matching, max-flow or network optimization, see Almohamad et al., Amaldi et al. or Komodakis et al. [1, 2, 36], logic inference, see Makhortov et al. or Krishnan [37, 46] and the optimization of NP-hard problems, such as the traveling salesman problem by using iterative subtour elimination. Here an efficient branching of the MILP prevents exploring sub-optimal solution spaces. Paul [56] gives a comprehensive introduction to formulate logic calculus (and beyond) using

MILPs. It is also possible to optimize decision trees as proposed by Zhu et al. [70]; or support vector machines, see Nguyen et al. [55], using respective LP formulations.

2.3 Trained Neural Networks and MILP

Already trained neural networks and mixed integer linear programming have been brought successfully together in the past. Note, that this work is focussing on the direct optimization of the network weights and its parameters from training data with a close connection to the works presented in Section 2.5. But since the combination of trained neural networks with MILP achieved very important results on network validation, verification and adversarial robustness, the following subsection summarizes some recent works in this field.

Neural networks and (mixed integer) linear programming have been brought together in the context of network fooling and model checking as proposed by Heo et al. and Modas et al. [31, 51]. Already trained networks can be analyzed using LPs, see Anderson et al. [3] and there exist works where neural networks are proposed to solve linear programs themselves, see Liu et al. [41]. Thiago et al. [64] introduce an approach to remove unneccessary units and layers of a neural network while not changing its output using a MILP. It therefore allows lossless compression of an existing network. Fischetti et al. [20] model a pertained DNN as a 0-1 Mixed Integer Linear Program (0-1 MILP) where the continuous variables correspond to the output values of each unit and a binary variable is associated with each rectified linear unit (ReLU). The paper discusses the peculiarity of such 0-1 MILP models and presents a bound-tightening technique to ease its optimization, e.g., for feature visualization and in the construction of adversarial examples. Grimstad et al. [23] consider the embedding of piecewise-linear deep neural networks (ReLU networks) as surrogate models in mixed integer linear programming (MILP) problems. Similar to other works, the formulation is obtained by programming each ReLU operator with a binary variable and applying the big-M method. As the efficiency of the formulation hinges on the tightness of the bounds defined by the big-M values, the presence of output bounds can be exploited in bound tightening. To this end, the authors present several bound tightening procedures that consider both input and output bounds. In Schweidtmann et al. [61], the training of neural networks and machine learning (ML) models are regarded as optimization problem where model parameters are varied to minimize the model error on the training data. Then, a subsequent optimization problem can be solved to identify the experimental conditions (i.e., inputs of the ML model) that maximize the experimental performance (i.e., the output of the ML model). This task is a subsequent optimization problem where the optimization contains an already trained ML model. In [66], the authors Tjandraatmadja et al. improve the effectiveness of propagation- and linear-optimization-based neural network verification algorithms by proposing a tightened convex relaxation for ReLU neurons. Based on this relaxation, the authors present two polynomial-time algorithms for neural network verification.

2.4 Software

In the recent years, several highly efficient software packages for optimization and machine learning have been proposed. The software package JANOS, proposed by Bergmann et al. [10], is a framework for optimization of variables under constraints. It supports linear regression, logistic regression and neural networks with rectified linear activation functions. The optimizer can only act on already trained neural networks. The optimization and machine learning toolkit (OMLT) introduced by Ceccon et al. in [14] is an open-source software package which also works for already trained neural network and gradient-boosted tree surrogate models. This package allows to make use of these models for larger optimization problems with applications, e.g., on maximizing a neural acquisition function or verifying neural networks. With reluMip, see Lueg et al. [44], it is possible to generate MILP models of trained artificial neural networks (ANNs) using the rectified linear unit (ReLU) activation function. At the moment, only TensorFlow sequential models are supported. OptiCL, see Maragno et al. [45], is an end-to-end framework for mixed integer optimization (MIO) with data-driven learned constraints. It establishes a methodological foundation for mixed integer optimization with learned constraints. The software proposes an end-to-end pipeline for data-driven decision making in which constraints and objectives are directly learned from data using machine learning (e.g., multi-layer perceptrons). Afterwards, the trained models are embedded in an optimization formulation. Finally it allows the capture of various underlying relationships between decisions, contextual variables, and outcomes. Scikit-learn [57] is a well known Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised ML problems. It supports tools for classification, regression, clustering, dimensionality reduction, model selection and preprocessing. It is based on standard methods of optimization, e.g., back-propagation-based neural network training based on autodiff.

Note, that most of these packages focus on the user interface and the optimization pipelines. These packages mainly support pre-trained networks and MILPs and do not focus on resource optimized models, as proposed in this work.

2.5 Neural Networks and MILP

The formulation of a linear threshold unit (LTU), which is similar to the perceptron described above, as an LP has been presented by Mangasarian already in 1993 [47]. Based on this formulation, several LTUs are iteratively connected using a so-called multisurface method tree. A significant difference to this work is, that the complete network as one MILP is optimized. The proposed model is not optimized in an iterative fashion. The special case of binary neural networks using MILP has been presented by Icarte et al. in [33]. Interestingly, binarized neural networks (BNNs) which are neural networks with weights and activations in $(-1, 1)$ not only can gain comparable test performance to standard neural networks, but allow for highly efficient implementations on resource limited systems as shown by Hubara et al. in 2016 [32]. The work by Bienstock et al. [11] establishes a general framework to reformulate (regularized) empirical risk minimization problems in machine learning into approximate

linear programs with explicit bounds on their complexity. The work of Kronqvist et al. [38] presents relaxations in between the big-M and convex hull formulations of disjunctions and draws advantages from both. The authors propose so-called *P-split* formulations to split convex additively separable constraints into P partitions and form the convex hull of the partitioned disjuncts. Experiments are performed on K-means clustering and neural networks with ReLU activation function. As the intermediate P-split formulations can form strong outer approximations of the convex hull with fewer variables and constraints than the extended convex hull formulations it can give significant computational advantages. Lu et al. [43] analyze the expressive power of neural networks which is important for understanding deep learning. The authors study how width affects the expressiveness of neural networks. Based on the knowledge that depth-bounded (e.g., depth-2) networks with suitable activation functions are universal approximators, the authors show a universal approximation theorem for width-bounded ReLU networks. Already width-$(n + 4)$ ReLU networks, where n is the input dimension, are universal approximators. In [65] Thorbjarnarson et al. use a mixed integer linear program to optimize neural network weights. There, also the optimization on the amount of neurons has been presented (and named as *model compression*). In contrast, this work additionally optimizes the sparsity on model weights, integer weights and the amount of neural activations.

There are two arguments why a MILP solution may be disadvantageous, as stated by Gambella et al. [22] (a) scaling to large datasets is problematic, as the size of the generated equations scale with the size of the training set and (b) solutions with provable optimal training error can overfit, thus training data is perfectly explained, but test performance is much worse. This paper focusses on the second challenge by the proposed sparsity constraints and resource optimization. The first challenge is confirmed in the experiments and summarized, e.g., in Table 5.

2.6 Feature Selection

Feature selection, see e.g., Guyon et al. [25] involves the task of measuring contributions of individual input features to the performance of a supervised learning task. It is an important tool (among others) for explainable/interpretable AI see Wojciech et al. [67]. An example application for feature selection is to understand from genomics data which genes or gene combinations are likely to cause a specific disease.

Feature selection has a reasonably long history with several competitions, such as the NIPS feature selection challenge, see Guyon et al. [27]. At this time, Bayesian networks and decision trees produced state-of-the art results. Due to the black-box behavior of neural networks, teaching a deep neural network to optimize on relevant features for decision making is quite challenging, see Romero et al. [59]. The work by Wojtas et al. [68] proposes to address feature importance ranking by using a dual-net architecture consisting of an operator and selector network. Ye et al. [69] propose collaborative feature selection based on an intermediate representation and subspace projection in the context of distributed learning. In [34] Ji et al. propose a particle swarm-based optimization for feature selection. The work from Ayindel et al. [5]

proposes networks with non-negative weights and demonstrates increased sparsity and competitive performance on many tasks.

In contrast to existing works, the proposed MILP formulation can easily respect constraints on sparsity. Additional constraints can further be used to enforce positive or even binary weights. Therefore, the proposed MILP formulation for optimizing weights of a neural network will allow an efficient feature selection during model optimization as shown in the experiments.

2.7 Resource Optimized Neural Networks

Optimizing neural network architectures means to optimize in a huge search space. Common approaches are based on reinforcement learning, structural search or performance prediction, as presented by Baker et.al, Cai et al. or Negrinho et al. [7, 12, 54]. Neural Architecture Search (NAS) is another common framework which automates the task of designing a neural network, especially the topology and the amount of used neurons across different layers, see Han et al. or Liu et al. [29, 40]. Still, all architecture search algorithms are computationally demanding despite their remarkable improvements over the last years. In this work a so-called *kill-switch* on neurons is proposed (see Fig. 4). Additionally, minimization on the neural activations is proposed. Both modifications fit well to the challenge of resource optimized neural networks [6, 9, 39, 53], as the resulting MILP formulation allows for optimizing a model to enforce sparsity on activations and additionally to minimize the amount of required neurons. As mentioned before, the constraints for minimizing the amount of neurons have also been proposed by Thorbjarnarson et al. [65] where a MILP is used for joint network training and compression, but only in combination with sparseness constraints on weights or activations (see also [19]) is the full potential exploited.

3 Modeling a Rosenblatt Perceptron as MILP

The Rosenblatt perceptron requires the computation of a scalar product, followed by a step function. This can also be expressed as an *if-else* condition and *greater as*, $>$, test. The formulations can be expressed as MILP using a so-called Big-M formulation. Note, that the letter M refers to a (sufficiently) large number associated with the artificial variables, see Paul [56].

The code fragment

```
if(δ) then c=d else c=e
```

can be expressed in an MILP as follows:

$$d - M(1-\delta) \leq c \leq d + M(1-\delta) \tag{3}$$

$$e - M\delta \leq c \leq e + M\delta \tag{4}$$

$$\delta \in \{0, 1\}. \tag{5}$$

The verification of this expression is quite simple:

$$
\begin{aligned}
\delta = 1 \rightarrow \quad & d - M(1-\delta) \leq c \leq d + M(1-\delta) \\
& \leftrightarrow d \leq c \leq d \\
& \rightarrow c \quad = \quad d \\
& \text{and} \quad e - M\delta \leq c \leq e + M\delta \quad \text{(trivial)} \\
\delta = 0 \rightarrow \quad & d - M(1-\delta) \leq c \leq d + M(1-\delta) \\
& \leftrightarrow d - M \leq c \leq d + M \quad \text{(trivial)} \\
& \text{and} \quad e - M\delta \leq c \leq e + M\delta \\
& \leftrightarrow e \leq c \leq e \\
& \rightarrow c \quad = \quad e.
\end{aligned}
$$

The condition

$$\texttt{a} > \texttt{b} \leftrightarrow \delta = \texttt{1}$$

can be expressed as

$$a \geq b + \epsilon - M(1-\delta) \tag{6}$$
$$a \leq b + M\delta \tag{7}$$
$$\delta \in \{0, 1\}. \tag{8}$$

The verification of this expression can be done in a similar fashion to equations (3)-(5). Based on equation (1), both expressions can be put together for modeling the activation of a perceptron with weights ω, bias b and input x as

$$\texttt{if} \left(\left(\omega^T x + b \right) > 0 \right) \texttt{ then 1 else 0}. \tag{9}$$

Please note, that the outcome is a binary vector which is later denoted as st_i. It can be interpreted as an input impulse of a transistor which is switching *on/off* a memorized value as input to the next layer, as visualized in Fig. 1.

Thus, for an output layer, the matrix vector product of the weights with the input reduces to an addition operation of memorized values so that the next perceptron can be expressed as MILP as well,

$$\texttt{if} \left(\left(\sum_{\text{st}_i = 1} q_i + b_F \right) > 0 \right) \texttt{ then 1 else 0}. \tag{10}$$

Please note, that such a perceptron can be interpreted as a neuron which performs neural memory allocation based on memorized values (q_i), see Kastellakis et al. [35].

As classification loss function for the MILP formulated neural network, the xor-error of the network outcome to each dimension of the one-hot encoded target vector is used. The xor-function can be modeled as MILP using

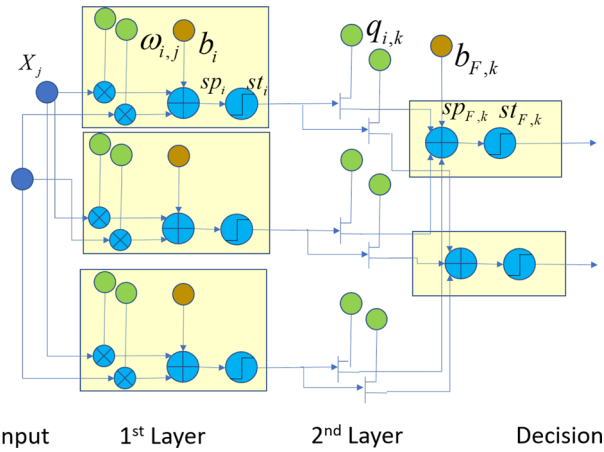

Fig. 1 Visualization of the neural network using separate building blocks of scalar products, summation operations, step functions and switches. This example network consists of two input values (X_1, X_2), three hidden neurons (marked as a yellow box) with weight matrix ($\omega_{i,j}$) and bias b_i to generate the scalar product value sp_i followed by a step function st_i. For the next layer, three switches pass over memorized values to a final layer with (in this case) two summation and step function operations for a 2-dimensional binary decision output

$$z \leq x + y \tag{11}$$
$$z \geq x - y \tag{12}$$
$$z \geq y - x \tag{13}$$
$$z \leq 2 - x - y \tag{14}$$
$$x, y, z \in \{0, 1\}. \tag{15}$$

Taking the input and output pair of the vector x and the one-hot encoded target y, a set of equations simulating the forward path for every training example is generated.

3.1 A Neural Network as MILP

Figure 1 visualizes the different parts required to formulate a training sample for a classification task as constraints for an MILP: The input is a m-dimensional input vector $X = (X_1, X_2, \ldots X_m)^T$ followed by n hidden neurons and a K-dimensional output. In the example illustration in Fig. 1, the input dimension m is set to two, the amount of neurons in the hidden layer n is set to three and the output dimension K is also set to two. The connection of the input vector to the hidden layer is a matrix–vector product involving the weights ω_{ij}. The index i indicates the number of the neuron and j the vector dimension. Then the bias values b_i for each neuron are added to the product and stored in the three slack variables sp_i. Then the step function is evaluated and the 0/1 values stored in the slack variables st_i. Afterwards, the weights $q_{i,k}$ and the values of the step function st_i are evaluated and stored in a slack variable $p_{i,k}$. This can be seen as the input for the next layer, which is already the K-dimensional output

layer. In each output perceptron, the sum of $p_{i,k}$ is computed, the bias $b_{F,k}$ added and stored in $sp_{F,k}$. Then a step function is evaluated and stored in the variable $\mathrm{st}_{F,k}$ and finally the xor-value (E_r) to the ground truth (GT) is computed for each output value as loss.

Whereas the learning parameters ω_{ij}, the bias values b_i, $b_{F,k}$ and the weights $q_{i,k}$ are simultaneously accessed for all training examples, the slack variables are unique for each training example used during optimization. The above steps generate a set of equality and inequality constraints, as well as integer conditions (for the step functions) which can be more concretely formalized by using the above MILP-formulations:

$$\min f^T x \ s.t. \tag{16}$$

Scalarproduct

$$\forall i = 1 \ldots n \qquad (17-23)$$

$$\sum_{j=1}^{m} X_j \omega_{ij} + b_i - sp_i = 0 \tag{17}$$

boolean var for step function

$$\epsilon - M(1 - \mathrm{st}_i) - sp_i \leq 0 \tag{18}$$

$$sp_i - M\mathrm{st}_i \leq 0 \tag{19}$$

$$(\forall k = 1 \ldots K)$$

if-else condition

$$q_{i,k} - M(1 - \mathrm{st}_i) - p_{i,k} \leq 0 \tag{20}$$

$$p_{i,k} - q_{i,k} - M(1 - \mathrm{st}_i) \leq 0 \tag{21}$$

$$-M\mathrm{st}_i - p_{i,k} \leq 0 \tag{22}$$

$$p_{i,k} - M\mathrm{st}_i \leq 0 \tag{23}$$

summation 2nd layer

$$\sum_{i=1}^{n} p_{i,k} + b_{F,k} - sp_{F,k} = 0 \tag{24}$$

Step Function (classification task)

$$\epsilon - M(1 - \mathrm{st}_{F,k}) - sp_{F,k} \leq 0 \tag{25}$$

$$sp_{F,k} - M\mathrm{st}_{F,k} \leq 0 \tag{26}$$

xor (classification task)

$$E_{r,k} - \mathrm{st}_{F,k} - \mathrm{GT}_k \leq 0 \tag{27}$$

$$-E_{r,k} + \mathrm{st}_{F,k} - \mathrm{GT}_k \leq 0 \tag{28}$$

$$-E_{r,k} + \mathrm{GT}_k - \mathrm{st}_{F,k} \leq 0 \tag{29}$$

$$E_{r,k} - 2 + \mathrm{st}_{F,k} + \mathrm{GT}_k \leq 0 \tag{30}$$

classification integer conditions

$$\mathrm{GT}_k, E_{r,k}, \mathrm{st}_{F,k}, \mathrm{st}_i \in \{0, 1\}. \tag{31}$$

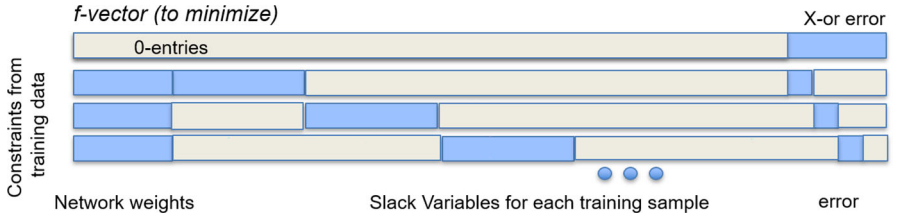

Fig. 2 Visualization of the target vector f which encodes the xor-error and the equality and inequality constraints for the respective network weights and slack variables for the training data

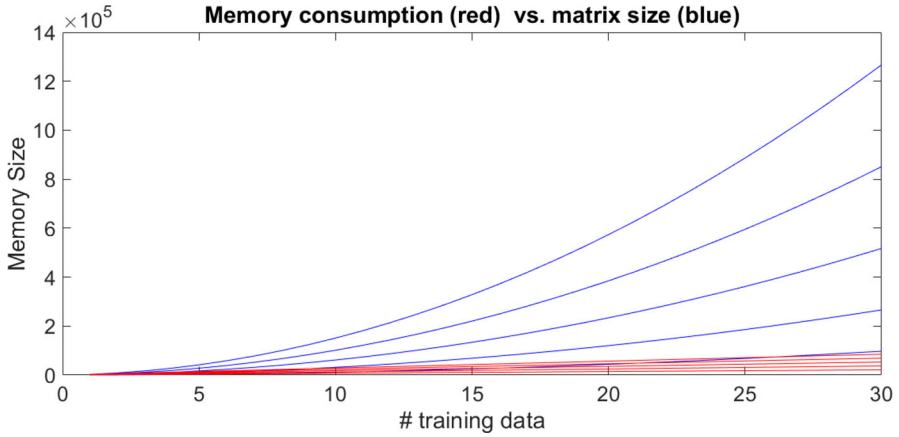

Fig. 3 Size of the equality and inequality constraint matrices. The x-axis gives the number of sample points (training data). The five curves show the increase for increasing numbers of neurons (from 1 to 5). The blue curve shows the size of the matrices, whereas the red curve displays the memory consumption for the variables. Even though the size is increasing nonlinearly, due to the use of sparse memory representations for the matrices, the increase in memory is nearly linear. (Best viewed in color and zoom in)

All variables can be ordered as a large vector and accessed via an index-operation ind(.). The equations lead to sparse vector expressions and describe an integer linear program which can be optimized with standard tools, e.g., based on Gurobi [24].

Figure 2 summarizes the resulting structure of the target vector f, the network parameters and the slack variables for each training sample on a classification task (using the xor-error). Minimizing the objective function f means to minimize the xor-error for all training examples and therefore the estimation of appropriate neural network weights. Thus, f is a vector containing only 0-values, except for the index holding the error $E_{r,k}$, e.g., $f(\text{ind}(E_{r,k})) = 1$. After optimization, the weights can be assembled to a neural network for forward inference on test data.

Figure 3 shows the size of the equality and inequality constraint matrices. The x-axis gives the number of sample points (training data). The five curves show the increase for increasing numbers of neurons (from 1 to 5). In this experiment, the dimension of the input and output is kept constant (to the value 2). These parameters also add up to the behavior of the curves for more complex settings. The blue curve displays the size of the matrices, whereas the red curve shows the memory consumption for the

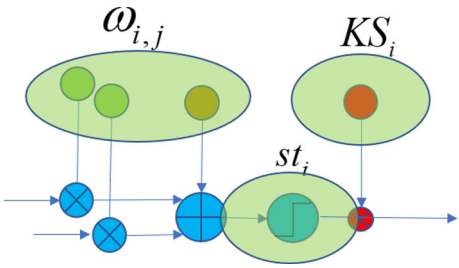

Fig. 4 Visualization of the different blocks to enforce sparsity: Sparsity can be enforced for the network weights $\omega_{i,j}$, the amount of activations can be minimized st_i or neurons can be completely switched off by adding a switch KS_i

variables. Even though the size is increasing polynomially, due to the use of sparse memory representations for matrices, the increase in memory is nearly linear. Note, that the amount of training data and the required slack variables can be a severe memory limitation for large sized problems. In the experiments, see Table 5, already smaller sized datasets lead to optimization problems of noticeable dimension, e.g., the ovarian dataset of dimension 100×4000 results for a neural network with up to 8 neurons in an inequality matrix A of size 73.148×68.172. Thus, it is mandatory to use sparse memory representations for both, the problem encoding and the optimization. Please also note, that it is also possible to generate deeper fully connected networks with this approach. As the equations are getting even larger, this will be investigated as part of future work.

3.2 Sparsity Constraints

As already discussed by Gambella et al. [22], a globally optimized model explaining the training data can perform bad on test data, due to overfitting. Thus, it is important to reduce the complexity of the model by enforcing sparsity on different involved components: Fig. 4 visualizes different parts of the neurons which can be modified together with the objective function to enforce sparsity: (1) Minimizing the weights $\omega_{i,j}$ and enforcing small, but positive integer values allows to optimize integer or binary networks. It requires to add $\omega_{i,j}$ to the objective function f and to enforce positive integer numbers on these weights. Additionally it is possible to limit the amount of weighting factors by adding an upper limit on the sum of weights $N_{\max}$. This allows, e.g., to enforce sparse binary weight matrices. The sparse matrices finally allow a semantic interpretation about relevant features for decision making. To balance the constraints and the overall error of the loss function, the scaling factor is empirically set to the value 0.1 for the experiments. This ensures the main focus is on the minimization of the classification error. Based on the earlier MILP, the following additional constraints can be added to enforce this property of sparse positive weight matrices:

$$f(\mathrm{ind}(\omega_{i,j})) = 0.1 \tag{32}$$

$$\sum \omega_{i,j} \leq N_{\max} \tag{33}$$

$$\omega_{i,j} \in \mathbb{N}^{+}. \tag{34}$$

As mentioned above, all variables used are assumed to be ordered and their positions are accessible via an index-operation ind(.).

(2) The second option to minimize the model is the *kill switch* (KS_i) which allows the minimization of the amount of neurons used. It is a simple if-else condition on the evaluated step function:

$$\mathrm{st}_i - M(1 - \mathrm{KS}_i) - p_i \leq 0 \tag{35}$$
$$p_i - \mathrm{st}_i - M(1 - \mathrm{KS}_i) \leq 0 \tag{36}$$
$$-M\mathrm{st}_i - p_i \leq 0 \tag{37}$$
$$p_i - M\mathrm{KS}_i \leq 0 \tag{38}$$
$$\mathrm{KS}_i \in \{0, 1\}. \tag{39}$$

The optimization function f needs to be modified with $f(\mathrm{ind}(\mathrm{KS}_i)) = 0.1$.

This variant for optimization has also been described by Thorbjarnarson et al. in [65] and called *model compression*.

(3) The final option for sparsity is minimizing the amount of activations (at the step functions st_i). It allows to minimize the neural effort for decision making. This is valid under the assumption, that the generation of a nervous activation is accompanied with a certain amount of energy. For this, the f-vector of the objective function needs to be modified by

$$f(\mathrm{ind}(\mathrm{st}_i)) = 0.1. \tag{40}$$

Thus, for example the sparsity of neural activations can be enforced by adding a small positive value to the objection function f at the positions of the evaluated step-functions on the *if-else* conditions st_i.

Even though the modifications appear simple and redundant, they have a major impact on the overall performance of the optimized models. Note, that the constraints are not exclusive, thus they can also be jointly optimized if desired. Even though they are simple to express as an MILP, they are very hard to optimize using differentiable programming. The effects of these different combinations of sparsity constraints are evaluated in detail in Section 4.3. The weighting factors (0.1) have been empirically chosen with the intention to have the main emphasis on the explanation of the training data. The optimization of the sparseness is important, but a subordinate task. The balancing factors across the sparsity variants can also be jointly optimized. This could be achieved, e.g., by using Bayesian optimization [62]. But it requires one to optimize and evaluate several MILPs, which can be very time consuming.

4 Experiments

Data loading and MILP formulation is implemented in MATLAB [48] and the integer linear program is optimized using Gurobi, see [24]. For this work, MATLAB version 2021b and the Gurobi version 9.1.2 build v.9.1.2rc0 (linux64) have been used. The optimization parameters for the MILP (presolving, rounding, diving, etc.) have not

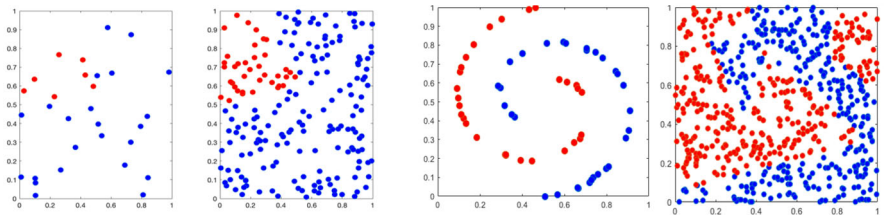

Fig. 5 Left: Training data and test for a simple corner dataset. Right: Training data and test for a spiral dataset

been changed throughout all experiments (default parameters are used). Thus, the same optimization problem leads to the same result, unless heuristics cause a different branching behavior. The optimizer stops once a global optimal solution is obtained or the maximum amount of computation time has been reached. This usually leads to a feasible solution which is accepted for evaluation. All results have been achieved in a Linux environment on Xeon Gold 5120 CPUs with 10 cores allocated for parallel computing. More recent software versions and better hardware will lead to a significant increase in performance. The experiments are divided in four parts. At first some results using standard datasets (e.g., from the UCI repository) and on synthetic data are presented. Then experiments on the feature selection challenge from NIPS 2003, which is available (at the time of writing) at CodaLab.org, are shown. Afterwards, detailed experiments on the proposed sparseness constraints are presented. Overall, experiments on more than 15 different datasets, including recent microarray datasets, are performed to cover a large variability of decision tasks.

4.1 Proof of Concept MILP

Figure 5 shows the training and testdata for a simple corner and spiral dataset. The respective networks take a 2-dimensional input and classify a binary output. The left example is based on two perceptrons in the hidden layer. Each perceptron learns a decision plane needed to correctly classify the upper left corner. The spiral dataset also shows that decision planes are learned, but here more planes are necessary to get a suited model after MILP optimization. Overall, this experiment shows, that the neural network formulated as an MILP can be optimized and that the weights of the shallow neural network learn effective decision planes to solve the optimization task.

4.2 Feature Selection Task

A useful property of the MILP formulation consists of the option to formulate additional constraints on the optimized weights. One constraint can be the restriction of the weights to binary values and a predefined maximal amount of features which are allowed to be used, see Section 3.2. A typical setting is the analysis of important features from highdimensional data for decision making, as it often occurs in medi-

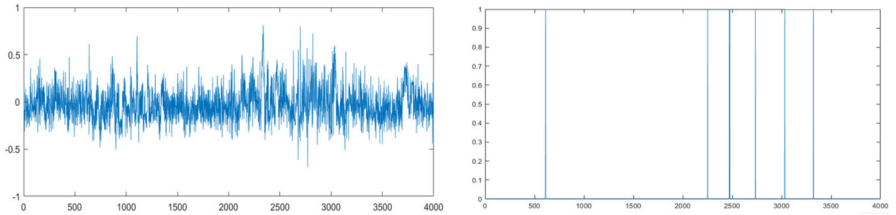

Fig. 6 Left: Features of the first hidden layer neuron after optimizing a shallow neural network. Right: Features of the MILP optimized neural network with sparsity constraints. Both networks are trained on 50% of the ovarian dataset and achieve an accuracy of 95% on the test set (best viewed in color and zoom in). The network on the right side is interpretable as selected factors are used for decision making

Table 1 Summary of the neurips feature selection benchmark dataset

Name	Size (MB)	Type	Features	# Train/Val	# Test
Arc.	8.7	Dense	10K	100/100	700
Gis.	22.5	Dense	5K	6000/1000	6500
Dex.	0.9	Sparse int.	20K	300/300	2000
Doro.	4.7	Sparse bin.	100K	800/350	800
Mad.	2.9	Dense	500	2000/600	1800

cal applications. The goal is to answer the question, which biological factors can be important for medical diagnostics or treatment.

For the next experiment the ovarian dataset [16] with 4000 features has been used. It consists of 200 examples, where 100 randomly selected examples are used for training and 100 for testing. Figure 6 shows the optimized weights of one standard perceptron of a shallow neural network which has been optimized using differential back propagation. As can be seen, many features are combined and it is not possible to tell which feature is relevant for decision making. Indeed, the risk for overfitting is very high. The image on the right shows the binary selected features of a globally optimized sparse neuron. This is much easier to interpret. Overall, the dataset is benign, both models achieve an accuracy of approx. 95% on the test set, but the overall quality heavily depends on the randomly selected samples. Please note, that the sparsity is crucial here, as shown in Table 3.

In 2003, NIPS hosted the feature selection challenge [26]. It consisted of five binary classification problems which are further explained and discussed in [28]. The challenge was the classification of high dimensional data with a limited amount of training data being available. Please note, that the challenge is still available at CodaLab.[2] Table 1 summarizes the dataset and its intrinsic properties.

As the benchmark is accessible at CodaLab, the MILP optimized neural networks have been used to participate in there. The proposed approach is currently listed on place 8, as shown in Table 2. For comparison, variants based on SVMs, decision trees and a random forest have also been evaluated and uploaded on the codalab server. Our proposed framework performs superior to these classical methods. Note,

[2] https://competitions.codalab.org/competitions/3931.

Table 2 Results of the proposed MILP net on the CodaLab benchmark and a comparison to three reference versions based on a random forest, svm and decision tree (accessed June 2022)

Rank	User	Arc.	Dexter	Doro.	Gis.	Mad.
8	**LP2NN**	0.84	0.91	0.70	0.968	0.87
10	RFBaseline	0.81	0.91	0.72	0.97	0.68
12	svmBaseline	0.86	0.96	0.64	0.92	0.51
13	DecTree	0.77	0.83	0.76	0.93	0.76

that the approach is in the top10 of this well established and frequently used dataset. Additionally, it is an outcome without any *bells-and-whistles*.

Please note, that this (automated) benchmark does only evaluate the accuracy on the test data. Thus, the initial goal of finding sparse solutions is not reflected in these scores. This would require one to balance and evaluate both aspects of sparsity and accuracy for different ML models which is a challenging task by itself. Therefore, this experiment only shows the possibility of optimizing non-differentiable neural network on various types of data. Table 2 shows the entry of the results obtained during the participation (with the username *LP2NN*). The MILP optimized neural networks are all very shallow, with up to 10 neurons. Tests with deeper nets and more neurons mainly increased the memory consumption and computation times without causing major changes in the scores. Better computational resources or more advanced optimization strategies may be able to change this in the future.

4.3 Sparse Models

In Section 3.2, three variants for enforcing sparsity on the model are proposed. Simple constraints in the MILP allow one to (a) reduce the entries of the weight matrices, (b) minimize the amount of neural activations and (c) reduce the amount of required neurons during training. A motivation is, that the complexity of a given optimization problem, and with this the amount of required neurons, is hard to estimate. One effect is, that performance is unsatisfactory without enough neurons, which could model the given task; or overfitting is observed, which means the model can fully explain the training data, but generalization fails.

For the experiments, the classical *wine*, *zoo*, *breastEW* and *ovarian* dataset have been used. The first two datasets are multicriterial classification tasks, with 3 categories for the wine dataset and 7 categories for the zoo dataset. The ovarian and breastEW dataset provides a binary test. For this experiment 50% of the data is used for training and 50% for testing and then each experiment is repeated ten times to analyze the obtained performance. For this experiment, the amount of neurons is set to the constant number 8. In Section 3.2 three variants to enforce sparsity are described, (1) by minimizing the weights, (2) by minimizing the amount of activations and (3) by minimizing the amount of neurons. All factors can be individually and jointly used in the MILP. Thus, 8 combinations are possible. The impact can be measured by several factors, (A) the accuracy on a test set (B) the amount of optimized weights (C) the amount of neuron activations and (D) the amount of neurons used. Table 3 summarizes the effect of switching off and on these individual factors. The numbers are the mean scores from 10 repetitions. For the accuracy, also the standard deviation of the results

Table 3 Evaluation of combining sparsity constraints during model optimization on the wine, breastEW, ovarian and zoo dataset: the letters b, c, d stand for the different constraints to minimize (b) the weights, (c) the amount of activations and (d) the amount of neurons

Flags	**wine**				**breastEW**			
(b,c,d)	Test (A) ↑	#ω (B) ↓	#act. (C) ↓	#neur (D) ↓	Test (A) ↑	#ω (B) ↓	#act. (C) ↓	#neur (D)
0,0,0	0.77± 0.09	47	511	8	0.80± 0.03	123	241	8
0,0,1	0.72± 0.06	43	497	2	0.87± 0.027	94.5	246	1.6
0,1,0	0.75± 0.05	46	53	8	0.79± 0.056	119.4	177	8
0,1,1	0.83± 0.09	44	53	2	0.85± 0.058	118	176	2.3
1,0,0	0.90± 0.06	17	448	6.4	0.92± 0.014	28	839	6.8
1,0,1	0.91± 0.02	11	434	2.2	0.89± 0.01	19	304	3
1,1,0	0.88± 0.05	13	71	5	**0.93**± 0.011	15	118	3.8
1,1,1	0.84± 0.03	12	71	3.7	0.91± 0.03	21	49	1
int 1,1,1	**0.94**± 0.03	10	91	2	0.90 ± 0.03	27	285	2.4
Flags	**ovarian**				**zoo**			
(b,c,d)	Test (A) ↑	#ω (B) ↓	#act. (C) ↓	#neur (D) ↓	Test (A) ↑	#ω (B) ↓	#act. (C) ↓	#neur (D)
0,0,0	0.53± 0.047	15893	417	8	0.72± 0.09	62	261	8
0,0,1	0.51± 0.06	15634	411	1	0.63± 0.06	60	244	3
0,1,0	0.54± 0.06	15328	357	8	0.74± 0.05	59	46	8
0,1,1	0.52± 0.07	15437	358	1	0.71± 0.09	49	55	5
1,0,0	0.82± 0.05	46	189	7.9	0.87± 0.06	17	211	6.4
1,0,1	0.82± 0.1	23	162	1	0.72± 0.02	25	282	3.1
1,1,0	0.77± 0.05	47	52	8	0.87± 0.05	18	43	7.3
1,1,1	0.91± 0.03	21	49	1	**0.91**± 0.03	15	46	5.6
int 1,1,1	**0.93** ± 0.03	20	105	2	0.90± 0.03	14	52	5.2

The impact is measured by (A) the accuracy on a test set (B) the weight sum (C) the absolute amount of neuron activations and (D) the final amount of used neurons
The row **int** denotes the performance of a discrete neural network with integer input and integer weights (best viewed in color)

is shown. Note, that the absolute numbers are not that relevant, as they depend on the size of the input vectors and the size of the network. Only the relative changes are important. It is clearly visible, that the constraints are working appropriately and they allow a large reduction in the complexity of the neural network, while maintaining (or even improving) the test accuracy. The color code in Table 3 indicates the respective sparsity constraint which has been activated during optimization. As the flag b indicates to enforce sparsity on the amount of neurons, the values in column #ω should significantly decrease when this bin is active. The same holds for flag c which penalizes the amount of activations #act and flag d which penalizes the amount of used neurons #$neur$. Note, that the factors can also be individually weighted, but for this experiment they were kept constant with a factor of 0.1 as mentioned in Section 3.2. It is also worth to notice that without the proposed sparsity constraints (where the flags are set to 0, 0, 0), the test performance is not competitive. The variant 0, 0, 1 corresponds to the model proposed by Thorbjarnarson et al. [65]. The results are inline with the comments of Gambella, et al [22], where it is shown that only minimizing the objective function (and thus to explain the training data) can lead to suboptimal generalization performance. Only with addition of the constraints, does suitable generalization occur. Overall, it also shows that many neural network models which are currently trained are over parameterized.

The numbers in Table 3 indicate that the sparsity on the network weights is the most important ingredient to improve generalization. But still, the optimization of the amount of activations and the minimization of the neurons are also important factors for an efficient and generalizing system. The row **int** denotes the performance of a discrete neural network with integer input and integer weights. To obtain integer input values each input dimension is scaled to [0, 1], multiplied by 10 and rounded. The optimizer is furthermore only optimizing integer weights for the weight matrices and bias vectors. Thus, together with the optimized step function, a network model is optimized which can work highly efficient on resource limited systems. Please note, that optimizing integer weights, and at the same time minimizing three other aspects (number of neurons, number of weights, and number of activations) is a highly challenging task for differentiable programming.

Table 4 summarizes the mean as well as the minimum and maximum computation times in seconds. The maximum is set to 12000 s. The computation time mainly depends on the complexity of the task with the size of input and output dimensions and the amount of examples, as well as the amount of required integer variables. The variance in the computation time can vary to quite some extent, but also after reaching the time limit, feasible solutions are obtained.

Table 5 summarizes the complexity of the obtained MILPs for different datasets. The column *Train* summarizes the dimensions of the training data in terms of $\#examples \times \#dimensions$, the dimensions of the obtained objective function f, the inequality constraints A and equality constraints Aeq. The last column *Encode* shows the time to generate the equations for the MILP solver in seconds. Note, that the equations are only generated once before the optimization. As all data is optimized simultaneously there are no iterations involved. Note, that a comparison of Tables 4 and 5 reveals that the dimensionality of the training data does not directly correlate to the required amount of time for optimization. Whereas the ovarian dataset is of reasonable size and converges after 2654 s for the binary flags (1, 1, 1), the optimization of the much smaller wine dataset takes up the maximum amount of time (12.000 sec). Even though after 12.000 s the performance gap with 0.1% is very small and the feasible solution of good quality. But still, the algorithm has not fully converged up till then. With respect to the hyper-parameters of the MILP optimizer, Gurobi parameters such as *Heuristics* or *MIPFocus* can speed up finding feasible solutions, but proving optimality still takes its time.

As a final experiment for challenging state-of-the-art classification tasks we perform an analysis on three gene expression DNA microarray datasets and follow the general protocol as described by Liu et al. in [42]. The properties of the dataset used are summarized in the left of Table 6. As the LP easily finds perfect solutions to model the training data, sparsity is necessary. The results and a comparison to state of the art [42], where intrinsically sparse networks are optimized using evolutionary deep learning, as well as another commonly used approach published by Mocanu et al. [50], is shown in the right of Table 6. To demonstrate the impact of the multicriterial sparsity of network weights and activations, the method proposed by Thorbjarnarson et al. in [65], where only the amount of neurons is minimized, is also shown. The decreasing performance shows that it is crucial to not only optimize the amount of neurons, but also the activations and the amount of weights.

Table 4 Mean and min/max computation times (in seconds) for different sparsity configurations on the wine, breastEW, ovarian and zoo dataset

Binary Flags (b,c,d)	Wine	BreastEW	Ovarian	Zoo
	seconds (mean, min–max)			
0,0,0	17	27	341	38
	10–28	12–79	126–694	24–55
0,0,1	29	189	129	409
	12–66	3–607	100–166	104–1749
0,1,0	63	293	214	50
	22–109	182–472	126–749	31–91
0,1,1	387	919	1419	4894
	180–12K	283–1791	539–3262	633–12K
1,0,0	57	8059	604	38
	17–87	288–12K	193–1084	11–74
1,0,1	12K	2467	473	3932
	50–12K	12K–12K	267–867	109–12K
1,1,0	981	12K	892	348
	3686–12K	12K–12K	380–1822	90–846
1,1,1	12K	12K	2654	5967
	12K–12K	12K–12K	987–3581	878–12K

Bin (b) indicates the minimization of the weights, bin (c) the minimization of the amount of neuron activations and bin (d) the minimization of the amount of used neurons, similar to Table 3

Table 5 Size of the resulting MILP for the used datasets

DataSet	Train	f	A	Aeq	Encode (sec)
Wine	89×13	1×4655	10804×4655	1246×4655	0.79
BreastEW	284×30	1×11340	24412×11340	3408×11340	4.02
Ovarian	108×4000	1×68172	73148×68172	1296×68172	149, 67
Zoo	50×16	1×5064	13122×5064	1100×5064	1.04

Shown is the input size of the training data (e.g., 89 examples with a 13 dimensional feature vector is used for training of the wine dataset), the size of the objective function f and the inequality and equality constraint matrices A and Aeq. The last column shows the time to generate the equations for the MILP solver in seconds

Table 6 Used microarray datasets and a comparison of the obtained results to alternative approaches

Dataset	Samples	Features	Classes	Set-MLP [42] (%)	MLP [50] (%)	Compression [65] (%)	Ours (%)
SUB-111 [30]	111	11.340	3	81.6	71.3	70.0	73
CAN-187 [63]	187	19.993	2	75.2	68.5	68	72
GLI-85 [21]	85	22.283	2	94.48	92.41	85	93

It is well known that these datasets seriously suffer from an extremely small number of samples which explains the high deviations among several solutions and makes it mandatory to enforce sparse solutions.

5 Summary and Discussion

Based on current developments in training NNs by using an exact optimization solver, this work presents an approach to optimize parameters of a Rosenblatt perceptron, and based on this the parameters of a neural network, as a MILP. The methodology followed in this paper is fundamentally different to differentiable programming and ensures a solution which is globally optimal (with respect to the loss function). The optimized weights can then be used to assemble a neural network for inference on unseen data. The parsing and definition of the MILP have been implemented in the MATLAB [48] environment and Gurobi [24] is used for optimization. In this paper, three constraints on sparsity are proposed to (a) minimize the network weights, to (b) minimize the amount of activations and to (c) minimize the amount of neurons. Additionally it is possible to optimize discrete models (only working on integer values). All constraints can be neatly formulated as additional constraints in the MILP and can even be jointly optimized. This allows sparsity and resource optimization of neural networks. Resource optimization involves the minimization of both, the amount of neurons, as well as the amount of activations required for decision making. An empirical comparison with the baseline formulation shows, that sparsity measures are important for generalization on unseen data.

The presented approach has one major drawback as mentioned earlier by Gambella et al. [22]: The computation time for optimization heavily increases with the amount of data and the amount of dimensions. Still, the used methodology can guarantee globally optimal results while being nearly parameter free. Especially, it is independent from random seeds, dropouts or other hyperparameters which can be crucial for high quality but are also time and energy demanding in differentiable programming. As part of basic and fundamental research, the used datasets are not in the same league of datasets used with large models and big data applications nowadays. Still, it is important to continue the research on alternative optimization strategies, especially on methods which can reach global optimality and methods that allow the imposition of conditions which are hard to optimize in differentiable programming.

Based on this work, several extensions are possible, ranging from modeling different layer types and operations such as convolutions, pooling or probabilistic models, see e.g., Capobianco et al. [13] or Awiszus et al. [4] to the optimization of the network topology, see Negrinho et al. [54]. It is also possible to generate deeper fully connected networks with the proposed approach. Here, an additional challenge is that the amount of equations and the amount of slack variables are heavily increasing. Recently, the use of graph neural networks for LP solving has been presented by Chen et al. [15]. Finally, only standard solvers are currently used. Intrinsic properties of the equality and inequality constraints should allow solving the system much faster, see Kronqvist et al. [38]. This will also allow to analyze and model deeper fully connected models as part of future work.

Acknowledgements The author received support from Leibniz Universität Hannover, Germany, and thanks the colleagues who provided the datasets used in this manuscript. All datasets are publicly available.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Almohamad, H.A., Duffuaa, S.O.: A linear programming approach for the weighted graph matching problem. IEEE Trans. Pattern Anal. Mach. Intell. **15**(5), 522–525 (1993)
2. Amaldi, E., Capone, A., Coniglio, S., Gianoli, L.G.: Network optimization problems subject to max-min fair flow allocation. IEEE Commun. Lett. **17**(7), 1463–1466 (2013)
3. Anderson, R., Huchette, J., Ma, W.: Strong mixed-integer programming formulations for trained neural networks. Math. Program. **183**, 3–39 (2020)
4. Awiszus, M., Rosenhahn, B.: Markov chain neural networks. In: IEEE Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 2180–2187, June (2018)
5. Ayinde, B.O., Zurada, J.M.: Deep learning of constrained autoencoders for enhanced understanding of data. IEEE Trans. Neural Netw. Learn. Syst. **29**(9), 3969–3979 (2018)
6. Bae, W., Lee, S., Lee, Y., Park, B., Chung, M., Jung, K.: Resource optimized neural architecture search for 3d medical image segmentation. In: Medical Image Computing and Computer Assisted Intervention - MICCAI 2019, pp. 228–236. Springer, Berlin (2019)
7. Baker, B., Gupta, O., Raskar, R., Naik, N.: Accelerating neural architecture search using performance prediction. In: 6th International conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Workshop track proceedings. OpenReview.net (2018)
8. Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.* **18**(1):5595–5637 (2017)
9. Bellec, G., Kappel, D., Maass W., Legenstein, R.: Deep rewiring: training very sparse deep networks. In: International Conference on Learning Representations (2018)
10. Bergman, D., Huang, T., Brooks, P., Lodi, A., Raghunathan, A.U.: Janos: an integrated predictive and prescriptive modeling framework. INFORMS J. Comput. **34**(2), 807–816 (2022)
11. Bienstock, D., Muñoz G., Pokutta, S.: Principled deep neural network training through linear programming (2019)
12. Cai, H., Chen, T., Zhang, W., Yu, Y., Wang, J.: Efficient architecture search by network transformation. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, Feb 2–7, 2018, pp. 2787–2794. AAAI Press (2018)
13. Capobianco, G., Cerrone, C., Di Placido, A., Durand, D., Pavone, L., Russo, D.D., Sebastiano, F.: Image convolution: a linear programming approach for filters design. Soft. Comput. **25**(14), 8941–8956 (2021)
14. Ceccon, F., Jalving, J., Haddad, J., Thebelt, A., Tsay, C., Laird, C.D., Misener, R.: Omlt: optimization and machine learning toolkit. arXiv: 2202.02414 (2022)
15. Chen, Z., Liu, J., Wang, X., Lu, J., Yin, W.: On representing linear programs by graph neural networks. arXiv: 2209.12288 (2022)
16. Conrads, T.P., Fusaro, V.A., Ross, S., Johann, D., Rajapakse, V., Hitt, B.A., Steinberg, S.M., Kohn, E.C., Fishman, D.A., Whitely, G., Barrett, J.C., Liotta, L.A., Petricoin, E.F., Veenstra, T.D.: High-resolution serum proteomic features for ovarian cancer detection. Endocr. Relat. Cancer **11**(2), 163–178 (2004)

17. Dantzig, G.B.: Maximization of a linear function of variables subject to linear inequalities. Act. Anal. Prod. Alloc. **13**, 339–347 (1951)
18. Dennis, J.E., Schnabel, R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Society for Industrial and Applied Mathematics (1996)
19. Ding T.W., Chin, R., Liu, Z., Marculescu, D.: Regularizing activation distribution for training binarized deep networks. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 11400–11409 (2019)
20. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints **23**, 296–309 (2018)
21. Freije, W.A., Castro-Vargas, F.E., Fang, Z., Horvath, S., Cloughesy, T., Liau, L.M., Mischel, P.S., Nelson, S.F.: Gene expression profiling of gliomas strongly predicts survival. Can. Res. **64**(18), 6503–6510 (2004)
22. Gambella, C., Ghaddar, B., Naoum-Sawaya, J.: Optimization problems for machine learning: a survey. Eur. J. Oper. Res. **290**(3), 807–828 (2021)
23. Grimstad, B., Andersson, H.: Relu networks as surrogate models in mixed-integer linear programs. Comput. Chem. Eng. **131**, 106580 (2019)
24. LLC Gurobi Optimization. Gurobi optimizer reference manual (2021)
25. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
26. Guyon, I., Gunn, S., Ben-Hur, A., Dror, G.: Result analysis of the nips 2003 feature selection challenge. In: Advances in Neural Information Processing Systems, vol. 17. MIT Press (2005)
27. Guyon, I., Gunn, S., Hur, A.B., Dror, G.: Result analysis of the nips 2003 feature selection challenge. In: Proceedings of the 17th International Conference on Neural Information Processing Systems, NIPS'04, pp. 545–552, Cambridge, MA, USA. MIT Press (2004)
28. Guyon, I., Nikravesh, M., Gunn, S., Zadeh, L.: Feature Extraction-Foundations and Applications, Studies in Fuzziness and Soft Computing, vol. 207. Springer, Berlin (2006)
29. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, NIPS'15, vol. 1, pp. 1135-1143. MIT Press, Cambridge (2015)
30. Haslinger, C., Schweifer, N., Stilgenbauer, S., Döhner, H., Lichter, P., Kraut, N., Stratowa, C., Abseher, R.: Microarray gene expression profiling of b-cell chronic lymphocytic leukemia subgroups defined by genomic aberrations and vh mutation status. J. Clin. Oncol. **22**(19), 3937–3949 (2004)
31. Heo, J., Joo, S., Moon, T.: Fooling neural network interpretations via adversarial model manipulation. In: Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc. (2019)
32. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Advances in Neural Information Processing Systems, vol 29. Curran Associates, Inc. (2016)
33. Icarte, R.T., Illanes L., Castro, M.P., Ciré A., McIlraith, S.A., Beck, J.C.: Training binarized neural networks using MIP and CP. In *Principles and Practice of Constraint Programming*. Springer, LNCS 11802 (2019)
34. Ji, B., Lu, X., Sun, G., Zhang, W., Li, J., Xiao, Y.: Bio-inspired feature selection: an improved binary particle swarm optimization approach. IEEE Access **8**, 85989–86002 (2020)
35. Kastellakis, G., Poirazi, P.: Synaptic clustering and memory formation. Front. Mol. Neurosci. **12**, 300 (2019)
36. Komodakis, N., Tziritas, G.: Approximate labeling via graph cuts based on linear programming. IEEE Trans. Pattern Anal. Mach. Intell. **29**(8), 1436–1453 (2007)
37. Krishnan, R.: PDM: a knowledge-based tool for model construction. In: [1989] Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume III: Decision Support and Knowledge Based Systems Track, vol. 3, pp 467–474 (1989)
38. Kronqvist, J., Misener, R., Tsay, C.: Between steps: intermediate relaxations between big-m and convex hull formulations. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 299–314. Springer, Cham (2021)
39. Le Cun, Y., Denker J.S., Solla S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605. Morgan Kaufmann (1990)
40. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: International Conference on Learning Representations (2019)
41. Liu, Q., Wang, J., Duch, W., Girolami, M., Kaski, S.: A one-layer dual recurrent neural network with a heaviside step activation function for linear programming with its linear assignment application.

In: Artificial Neural Networks and Machine Learning—ICANN 2011, pp. 253–260. Springer, Berlin (2011)
42. Liu, S., Mocanu, D., Matavalam, A., Pei, Y., Pechenizkiy, M.: Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. Neural Comput. Appl. **33**, 04 (2021)
43. Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L.: The expressive power of neural networks: a view from the width. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, pp. 6232–6240. Curran Associates Inc, Red Hook (2017)
44. Lueg, L., Grimstad, B., Mitsos, A., Schweidtmann, A.M.: Relumip: open source tool for milp optimization of Relu neural networks (2021)
45. Donato, M., Holly W.: OptiCL: Mixed-integer optimization with constraint learning (2021). https://github.com/hwiberg/OptiCL/
46. Makhortov, S., Ivanov, I.: Equivalent transformation of the reasoning model in production zeroth-order logic. In: 2020 International Conference on Information Technology and Nanotechnology (ITNT), pp. 1–4 (2020)
47. Mangasarian, O.L.: Mathematical programming in neural networks. ORSA J. Comput. **5**, 349–360 (1993)
48. MATLAB. R2021b (2021)
49. McCulloch, W., Pitts, W.: A logical calculus of ideas immanent in nervous activity. Bull. Math. Biophys. **5**, 127–147 (1943)
50. Mocanu, D., Mocanu, E., Stone, P., Nguyen, P.H., Gibescu, M., Liotta, A.: Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nat. Commun. 9 (2018)
51. Modas, A., Moosavi-Dezfooli, S.-M., Frossard, P.: Sparsefool: a few pixels make a big difference. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9079–9088 (2019)
52. Murty, K.G.: Linear Programming. Wiley, Hoboken (1983)
53. Narang, S., Diamos, G., Sengupta, S., Elsen, E.: Exploring sparsity in recurrent neural networks. In: International Conference on Learning Representations (2017)
54. Negrinho, R., Patil, D., Le, N., Ferreira, D., Gormley, M., Gordon, G.: Towards modular and programmable architecture search. Neural Inf. Process. Syst. (2019)
55. Nguyen, H.T., Franke, K.: A general lp-norm support vector machine via mixed 0–1 programming. In: Machine Learning and Data Mining in Pattern Recognition, pp. 40–49. Springer, Berlin (2012)
56. Paul, W.H.: Logic and Integer Programming, 1st edn. Springer, Berlin (2009)
57. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
58. Picard, D.: Torch.manualseed(3407) is all you need: on the influence of random seeds in deep learning architectures for computer vision. arXiv: 2109.08203 (2021)
59. Romero, E., Sopena, J.: Performing feature selection with multilayer perceptrons. IEEE Trans. Neural Netw. **19**, 431–441 (2008)
60. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386–408 (1958)
61. Schweidtmann, A.M., Bongartz, D., Mitsos, A.: Optimization with trained machine learning models embedded. arXiv: 2207.12722 (2022)
62. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, pp. 2951–2959 (2012)
63. Spira, A., Beane, J.E., Shah, V., Steiling, K., Liu, G., Schembri, F., Gilman, S., Dumas, Y., Calner, P., Sebastiani, P., Sridhar, S., Beamis, J., Lamb, C., Anderson, T., Gerry, N., Keane, J., Lenburg, M.E., Brody, J.S.: Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer. Nat. Med. **13**(3), 361–366 (2007)
64. Thiago, S., Abhinav, K., Srikumar, R.: Lossless compression of deep neural networks. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 417–430. Springer, Cham (2020)
65. Thorbjarnarson, T., Yorke-Smith, N.: Optimal training of integer-valued neural networks with mixed integer programming. PLOS ONE **18**(2) (2023)
66. Tjandraatmadja, C., Anderson, R., Huchette, J., Ma, W., Patel, K.K., Vielma, J.P.: The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. In: Advances in Neural Information Processing Systems, vol. 33, pp. 21675–21686. Curran Associates, Inc. (2020)

67. Wojciech, S., Müller, K.-R.: Towards explainable artificial intelligence. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, Lecture Notes in Computer Science, vol. 11700, pp. 5–22 (2019)
68. Wojtas, M., Chen, K.: Feature importance ranking for deep learning. In: Advances in Neural Information Processing Systems, vol. 33, pp. 5105–5114. Curran Associates, Inc. (2020)
69. Ye, X., Li, H., Imakura, A., Sakurai, T.: Distributed collaborative feature selection based on intermediate representation. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI-19, pp. 4142–4149. International Joint Conferences on Artificial Intelligence Organization, vol. 7 (2019)
70. Zhu, H., Murali, P., Phan, D., Nguyen, L., Kalagnanam, J.: A scalable MIP-based method for learning optimal multivariate decision trees. In: Advances in Neural Information Processing Systems, vol. 33, pp. 1771–1781. Curran Associates, Inc. (2020)