ReLU Networks as Surrogate Models in Mixed-Integer Linear Programs

Bjarne Grimstad[a,*], Henrik Andersson[b]

[a]*Solution Seeker AS, Gaustadalléen 21, 0349 Oslo, Norway*
[b]*Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway*

Abstract

We consider the embedding of piecewise-linear deep neural networks (ReLU networks) as surrogate models in mixed-integer linear programming (MILP) problems. A MILP formulation of ReLU networks has recently been applied by many authors to probe for various model properties subject to input bounds. The formulation is obtained by programming each ReLU operator with a binary variable and applying the big-M method. The efficiency of the formulation hinges on the tightness of the bounds defined by the big-M values. When ReLU networks are embedded in a larger optimization problem, the presence of output bounds can be exploited in bound tightening. To this end, we devise and study several bound tightening procedures that consider both input and output bounds. Our numerical results show that bound tightening may reduce solution times considerably, and that small-sized ReLU networks are suitable as surrogate models in mixed-integer linear programs.

Keywords: Deep Neural Networks, ReLU Networks, Mixed-Integer Linear Programming, Surrogate Modeling, Regression

1. Introduction

Access to new and more data and the capabilities of modern machine learning algorithms have created vast possibilities for data-driven decision support systems within operations research and mathematical programming. We are now able to utilize the data and build highly complex models to describe nonlinear phenomena. A challenge in mathematical optimization is how to use these models in an optimization framework and how to efficiently solve the problems that arise.

One of the most promising techniques to tackle this challenge is the application of surrogate models as substitutes for nonlinear relationships. The importance of developing surrogate models is three-fold: 1) it allows for optimization of functions with an unknown/hidden mathematical structure; 2) it may reduce solution times when highly complex functions can be substituted by surrogate models with better properties for optimization (e.g. smoothness and linearity); and 3) for problems with unknown functions that are in some sense expensive to sample, surrogate models can be used to handle the trade-off between exploration and exploitation [1, 2].

Many of the classic function approximation methods have been applied to build surrogate models in process optimization [3, 4, 5, 6, 7]. Examples include polynomial tensor-product splines and radial basis functions, which result in smooth surrogate models that can be optimized by nonlinear (global) solvers. An important subset of surrogate models is the piecewise-linear (PWL) models, which naturally lend themselves to mixed-integer linear programming (MILP) formulations. Various MILP formulations exist for PWL models of one or multiple variables, including the ones listed in [8]. The starting point of these formulations is a set of sample points to be interpolated. As will be discussed subsequently, some of these formulations put restrictions on the sample structure in addition to the interpolation constraints, limiting their applicability to low-dimensional functions.

In many ways, it is more practical to build PWL surrogate models directly using regression methods. Practitioners can then leverage modeling tools for statistical analysis and machine learning. With appropriate formulations, the resulting regression models can be brought into a MILP framework for optimization. In this spirit, many works have been published recently on how to embed trained machine learning models in optimization [9, 10, 11, 12]. For example, [9] recently employed a MILP formulation

[*]Corresponding author
Email address: bjarne.grimstad@gmail.com (Bjarne Grimstad)

for multivariate adaptive regression splines (MARS), while [10] developed a framework for parameter prediction that incorporates the objective and constraints of the optimization problem explicitly.

We follow a similar path in this work by employing a MILP formulation for the general class of *ReLU networks*. These are networks composed of max-affine spline operators (piecewise-linear and convex operators), which include: affine operators, ReLU-type activations, absolute value activations, convolution operators, and max/mean pooling. By construction, ReLU networks are affine spline operators, which are piecewise linear, but not necessarily convex [13]. In this class of deep neural networks (DNNs) we find widely applied architectures such as CNNs, ResNets, inception networks, maxout networks, network-in-networks, scattering networks, and their variants using max-affine spline operators. These networks are used to model complex relationships and often achieve state-of-the-art performance in terms of modeling accuracy. In this work we consider ReLU networks composed solely of affine operators and ReLU activations, since these operators suffice for many function approximation tasks.

An exact MILP formulation can be obtained by programming each ReLU operator with a binary variable and applying the big-M method. This formulation has recently been applied to formal verification [14, 15, 16, 17, 18], to count linear regions [19], and to compress DNNs [20]. We use the same formulation to solve optimization problems with embedded ReLU networks. This allows us to leverage powerful modeling frameworks like TensorFlow [21] to build PWL surrogate models, and then optimize these using state-of-the-art MILP solvers.

The main focus and contributions of this paper is to show how ReLU networks can be used to model complex phenomena in a mixed-integer linear framework and how bound tightening affect the performance of the big-M formulation. The main contributions of the paper are summarized as follows:

– A discussion about the advantages of using ReLU networks when modeling complex phenomena in a MILP framework.

– A general framework for tightening bounds that unify different bound tightening procedures for output-bounded ReLU networks under one general procedure.

– A computational study showing the strength of the bound tightening procedures, but also the important trade-off between time spent on bound tightening and the solution time of the optimization problem.

– A real application where bound tightening makes the difference between not finding a feasible solution and solving the problem to optimality in short solution times.

1.1. Structure of the paper

We begin in Section 2 by discussing the various aspects of building piecewise-linear approximations of nonlinear functions, and embedding these as surrogate models in optimization. We highlight the advantages of modeling multi-variable functions via ReLU networks, as opposed to using interpolation on simplices, which has been the traditional approach in MILP literature. After presenting the MILP formulation in Section 3, we devise several optimization-based bound tightening procedures for output-bounded ReLU networks in Section 4. To study the procedures' efficiency, and the feasibility of using ReLU networks as surrogate models in process optimization, we present a computational study in Section 5. Our test suite includes a challenging oil production optimization problem, for which the complicated physics of multiphase flow in pipes is modeled by ten ReLU networks. Concluding remarks and promising research directions are given in Section 6.

2. Piecewise-linear approximations of nonlinear functions

We consider the modeling of a nonlinear function $f : D \subset \mathbb{R}^n \to \mathbb{R}$, on a compact domain $D \subset \mathbb{R}^n$ (we assume that D is a polytope). To incorporate f in a MILP model, it must be approximated by a piecewise-linear function. For cases where f is unknown, nonseparable (for $n \geq 2$), or highly complex, a general approach to the approximation of f is to sample it on the domain D, and then build a piecewise-linear approximation from the sample. In cases where f is not a mathematical construct, e.g. f could be a real process, the sample may represent a set of experiments. In general, a sample consists of non-structured sample points scattered in D.

A common restriction of many approximation methods is that f must be sampled on a rectilinear grid G, covering a hyper-rectangular domain D. A rectilinear grid on which variable x_i is partitioned into m_i intervals, results in $\prod_{i=1}^{n} m_i$ boxes (hyperrectangles). Clearly, the number of boxes grows exponentially with n. To obtain a piecewise linear model that interpolates the sample points, each of these boxes must be divided into a set of simplices. The minimum number of simplices needed to triangulate an n-dimension hypercube is 1, 2, 5, 16, 67, 308, and 1493 for $1 \leq n \leq 7$, respectively, see [22]. In general, an upper bound is given by $n!$. Thus, a partitioning of D may consist of up to $n! \prod_{i=1}^{n} m_i$ simplices.

Table 1 lists the lower bound on the number of simplices resulting from a rectilinear grid partitioning. The bounds are given for $1 \leq n \leq 7$, with each variable partitioned into ten intervals. Reading the table, we see that the exponential increase in the number of boxes (or simplices) prohibits any practical use of such a partitioning for $n > 3$. For example, [23] and [8] consider grids for $n \leq 3$.

Table 1: Number of simplices resulting from a rectilinear grid partitioning of the domain, where each variable is partitioned into ten intervals. We have used the lower bound on the number of simplices per box.

Dimension n	# boxes	# simplices/box	# simplices
1	10	1	10
2	100	2	200
3	1 000	5	5 000
4	10 000	16	160 000
5	100 000	67	6 700 000
6	1 000 000	308	308 000 000
7	10 000 000	1 493	14 930 000 000

Another disadvantage with rectilinear sampling, stemming from its structure, is its inability to *locally* control the sampling resolution. That is, increasing the sampling resolution in a region of D where f has important features (e.g. large gradients), will also increase it in other regions where it may not be necessary to sample densely. Rectilinear sampling is also subject to high sparsity of sample points in higher dimensions. Although this aspect of the "curse of dimensionality" applies to all sampling methods, it is becomes especially prominent for rectilinear sampling due to its inability to locally control the sampling resolution.

A common tactic to avoid the limitations of rectilinear sampling is to apply adaptive sampling, i.e. an algorithm that dynamically selects sample points to minimize the (expected) approximation error. This is particularly useful when f is expensive to evaluate. Various sampling strategies exist for which sample points are selected successively to reduce the approximation error, [24, 25]. Termination criteria may be on the total number of sample points or that the estimated approximation error is below some threshold, [26]. Some optimization algorithms include such a sampling in the optimization routine, in an attempt to balance between the optimization objective and lowering the approximation error.

Sampling algorithms like those described above typically result in sample points that are scattered in D without any particular structure. The same holds true for most samples obtained from real experiment data. A piecewise-linear model can be obtained from such samples via triangulation of the sample points, resulting in a simplex partitioning of D. There exist several MILP models for PWL functions that have no requirements on the family of polytopes/simplices produced by the triangulation, and which can be used with a triangulation scheme for scattered sample points. One example is the DLog model in [8], which uses a logarithmic number of binary variables to model a PWL function. This model has shown good computational performance for higher-dimensional functions ($n \geq 2$), compared to MILP models with a linear number of binary variables. However, as is shown subsequently, this modeling approach is not feasible for the oil production optimization case considered in this paper.

There exist MILP formulations for PWLs that achieve higher efficiency by putting requirements on the partitioning scheme. E.g., the Log model of [8] requires a triangulation scheme compatible with the J_1 ("Union Jack") triangulation, while SOS2 models, [27], require a rectilinear partitioning. In general, these highly structured partitioning schemes do not conform with a sample of scattered points and are also subject to the dimensionality issues discussed above.

So far we have only discussed approximation methods that interpolate the sample points.[1] If this requirement is relaxed, we may consider regression methods using piecewise-linear splines. These methods solve a regression problem by minimizing the approximation error on the sample. We may divide these methods into two classes: non-adaptive methods for which the domain partitioning is fixed (given by break-points or knots); and adaptive methods where the regression problem finds both a polytopic partition of the domain and the linear pieces in each polytope. MARS and ReLU networks are examples of adaptive methods.

[1]We note here that for $n \geq 2$, SOS2 models do not interpolate all sample points.

Regression models offer a nice alternative to the interpolating methods in that their complexity (number of linear pieces) is easily controlled. This is a key property when building predictive models, as it allows the modeller to strike a balance between underfitting and overfitting the data. Additionally, it may also be advantageously used to reduce unnecessary complexity of models, before attempting to use them as surrogates in a mathematical optimization problem. That is, the principle of *Occam's razor* is favorably applied in both the modeling and optimization setting.

2.1. ReLU networks as piecewise-linear surrogate models

Below, we highlight some important properties of ReLU networks that make them attractive as surrogate models in optimization.

Piecewise-linear and continuous. ReLU networks are composed of max-affine spline operators and are piecewise-linear and continuous functions. This means that a ReLU network can be explicitly formulated and optimized in a MILP framework. This is opposed networks composed of other operators, e.g. sigmoid or hyperbolic functions, which require piecewise-linear approximations.

Suited for modeling of high-dimensional and complex functions. It is easy to control the complexity of a ReLU network by adjusting its depth and width. As depth and width increases, so does the number of nodes (also called neurons) and effectively the expressiveness of the model. An upper bound on the number of linear regions generated by a neural network with L hidden layers of width W, and inputs $x \in \mathbb{R}^{n_0}$ is $\mathcal{O}(W^{Ln_0})$ [28]. For fixed W and n_0, this upper bound indicates that the number of regions grows exponentially in the network depth L. The bound shows that deep networks can be more expressive than shallow networks (can be seen by keeping WL constant and varying L). This observation has been used to partly explain the success of DNNs [19].

Adaptive partitioning of the domain. Changes in the parameters of a ReLU network automatically induce changes in the partition of the domain [13]. When the network parameters are updated during training, regions of the domain where the data has important features are automatically described with smaller polytopes, while other parts of the domain are described by larger polytopes. Figure 1 shows an example of a domain partition, which is refined from one layer to the next.

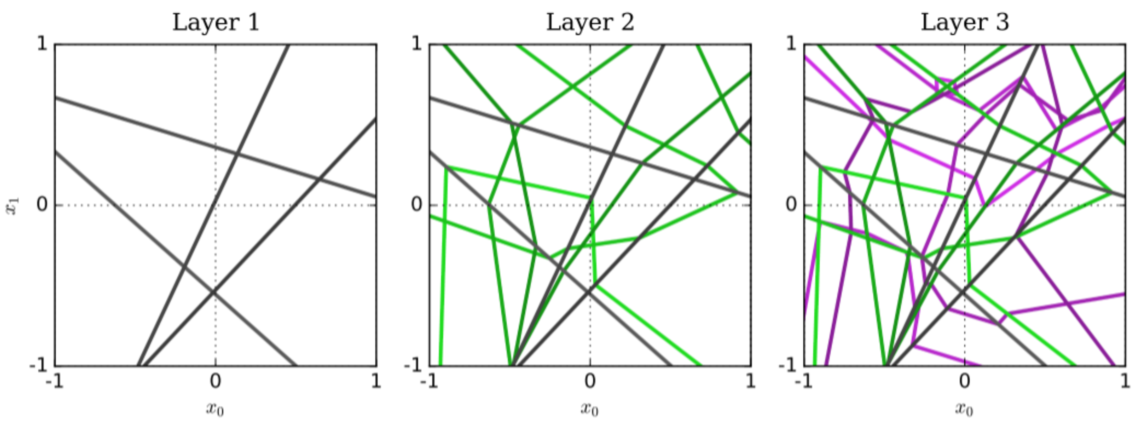


Figure 1: Example of domain partition generated by a multilayered neural network; adopted from [28].

Can be trained on scattered and noisy data. The process of training a ReLU network does not put requirements on the distribution of the data points and is thus compatible with any sampling algorithm. Furthermore, an arsenal of regularization techniques are available for neural networks in general. These techniques allow neural networks to generalize from noisy data. Scattered and noisy data are issues that occur in many practical applications.

Scale to large datasets. Neural networks are usually trained with stochastic or mini-batch gradient descent algorithms [29]. These algorithms optimize over the parameters Θ of a neural network f_Θ to find an approximation of some true function f. For regression tasks, it is common to minimize the empirical loss

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|y^{(i)} - f_\Theta(x^{(i)})\|_2^2 + \lambda \|\Theta\|_2^2,$$

where $\{x^{(i)}, y^{(i)}\}_{i=1}^N$ are N samples of f. The first term specifies the mean squared error, while the second is an L_2 regularization term that penalizes large values of Θ. The hyperparameter $\lambda \in \mathbb{R}_{\geq 0}$ is used to control model complexity and encourage good generalization.

During optimization, the parameters are updated using gradients $\partial \mathcal{L} / \partial \Theta$ computed using backward-propagation. The batch size N can be selected to control the computation and memory resources required to evaluate $\mathcal{L}$ and its gradients. This enables training of deep neural networks on large datasets with millions of data points.

Many software tools available for modeling. A plethora of software tools exists for designing, training and performing inference tasks with ReLU networks. In this work we use the machine learning framework TensorFlow [21].

3. A 0-1 MILP model for ReLU networks

We consider a ReLU network with $K+1$ layers, numbered from 0 to K. Layer 0 is the input layer (usually not counted as a layer), while the last layer K is the output layer. Each layer $k \in \{0, \dots, K\}$ has n_k nodes, numbered from 1 to n_k. For simplicity we consider networks of fully connected layers with parameters $\theta^k = (W^k, b^k)$ for $k \in \{1, \dots, K\}$, where $W^k \in \mathbb{R}^{n_k \times n_{k-1}}$ and $b^k \in \mathbb{R}^{n_k}$. We state the network as $f_\Theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_K}$, where K, n_0, and n_K are implied by the set of parameters $\Theta := \left\{\theta^1, \dots, \theta^K\right\}$.

Let $x^k \in \mathbb{R}^{n_k}$ be the output of layer k, and x_j^k the output of the j-th node for $j = 1, \dots, n_k$. Following this notation, x_j^0 is the j-th input value, and x_j^K is the j-th output value (out of n_K) of the network. For each (hidden) layer $k \in \{1, \dots, K-1\}$, the output vector x^k is computed as

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

where we denote by $\sigma(y) := \max\{0, y\}$ (componentwise) the ReLU activation function for a real vector y. The output of the network, $x^K \in \mathbb{R}^{n_K}$, is given by the affine equation $x^K = W^K x^{K-1} + b^K$.

The ReLU operator can be programmed in multiple ways. Following [18], we consider the linear equation

$$w^\mathsf{T} y + b = x - s, \tag{2}$$

where the output of the ReLU is decoupled into a positive part $x \geq 0$ and negative part $s \geq 0$. The output of the ReLU can then be obtained by imposing that at least one of the two terms x and s must be zero. Assuming that we may compute finite values L and U so that $L \leq w^\mathsf{T} y + b \leq U$, we may explicitly program the ReLU logic via big-M constraints:

$$\begin{aligned} & x \leq Uz \\ & s \leq -L(1-z) \\ & z \in \{0, 1\} \end{aligned} \tag{3}$$

where we have introduced a binary *activation variable* z.

Using a binary activation variable for each node (j, k), the network can be expressed with the following

0-1 MILP formulation:

$$
\begin{aligned}
&\text{Input layer} && && \\
&L^0 \leq x^0 \leq U^0, && && \text{(4a)}\\
&\text{Hidden (ReLU) layers} && && \\
&\left.\begin{aligned} &W^k x^{k-1} + b^k = x^k - s^k \\ &x^k, s^k \geq 0 \end{aligned}\right\} && k = 1, \dots, K-1, && \text{(4b)}\\
&z^k \in \{0,1\}^{n_k} && k = 1, \dots, K-1, && \text{(4c)}\\
&\left.\begin{aligned} &x^k \leq U^k z^k \\ &s^k \leq -L^k(1 - z^k) \end{aligned}\right\} && k = 1, \dots, K-1, && \text{(4d)}\\
&\text{Output layer} && && \\
&W^K x^{K-1} + b^K = x^K, && && \text{(4e)}\\
&L^K \leq x^K \leq U^K. && && \text{(4f)}
\end{aligned}
$$

This is an exact formulation of the ReLU network, meaning that the output of this formulation always is the same as the output from the ReLU network for the same input. That is, for a fixed input vector x^0, all the other variables in (4) are fixed, including the output variables x^K. The only "degenerate" solutions occur when the input to a node (j, k) is zero, in which case the value of z_j^k can arbitrarily be set to 0 or 1 without affecting the output.

There is one binary variable for each hidden node in the network, this means that the formulation scales linear with the number of hidden nodes. The constraint sets (4b) and (4e) can be written av $A_x x + A_s s = -b$, where A_x and A_s have a special block angular structure (depends on how we stack the variables x and s), see Figure 2. Unless sparsity is encouraged, for example by utilizing L_1 regularization, the parameters W^k and b^k are typically dense for a trained network.

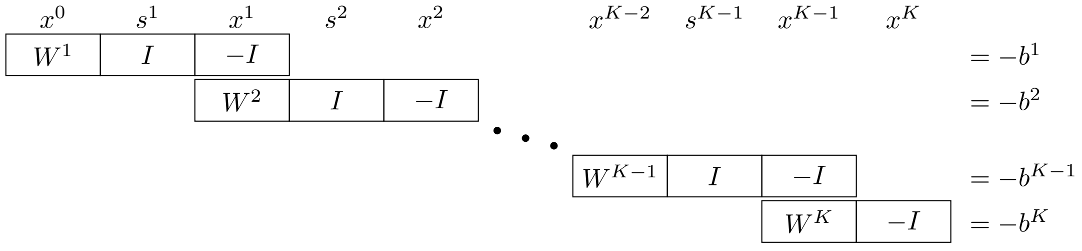


Figure 2: The block angular structure of the constraint set of a ReLU network

The tightness of an LP relaxation of (4) depends heavily on the size of the big-M values L^k and U^k, for layers $k = 1, \dots, K-1$. Weak LP relaxations result from large big-M values, and may severely hamper the efficiency of the solver. We explore several bound tightening procedures in the upcoming section. The computed bounds, L_j^k and U_j^k, constrain the variables as follows:

$$
\begin{aligned}
\max\{0, L_j^k\} \leq x_j^k \leq \max\{0, U_j^k\} \\
\max\{0, -U_j^k\} \leq s_j^k \leq \max\{0, -L_j^k\}
\end{aligned}
\tag{5}
$$

Before continuing, we remark that alternative formulations exist for the ReLU logic. The logic may for example be programmed via complementary constraints ($xs = 0$) or indicator constraints ($z = 0 \implies x \leq 0$ and $z = 1 \implies s \leq 0$). An advantage with these formulations is that they do not require explicit big-M values. However, these formulations tend to produce very hard mixed-integer instances that challenge state-of-the-art solvers [18].

4. Bound tightening procedures

We study procedures for computing valid bounds $B := \{B^0, \dots, B^K\}$, where $B^i := [L^i, U^i]$ for $i = 0, \dots, K$. We begin by defining the feasible set of a ReLU network f_Θ bounded by B as

$$
P(\Theta, B) := \{(x, s, z) : \text{(4) is satisfied}\}. \tag{6}
$$

The projection of this set onto (x^0, x^K), given by $F(\Theta, B) := \{(x^0, x^K) : (x, y, z) \in P(\Theta, B)\}$, yields all possible input-output pairs of f_Θ. The inputs x^0 and outputs x^K are directly constrained by the bounding boxes B^0 and B^K, respectively.

Now consider the additional constraints $x^0 \in D$ and $x^K \in E$, where $D \in \mathbb{R}^{n_0}$ and $E \in \mathbb{R}^{n_K}$ are polytopes. We then have that

$$\{(x^0, x^K) : x^0 \in D, x^K \in E, (x, y, z) \in F(\Theta, B)\} \subseteq F(\Theta, B) \tag{7}$$

The reason for making this, perhaps obvious point, is that a DNN programmed by (4) may be part of a larger optimization problem with additional constraints on x^0 and x^K. These constraints may tighten variable bounds and should be considered when performing bound tightening.

Previous works on bound tightening (BT) of (4) have not included output bounds E. The primary goal has been to search in $F(\Theta, B)$ subject only to input bounds D, which explains the focus on *forward-propagating* BT procedures. To find optimal bounds with the addition of output bounds E, it is required that the output bounds are propagated backwards through the DNN. In the following, we study several bound tightening procedures for (4) with D and E included. We will highlight the procedures that are capable of propagating bounds backwards and subsequently investigate the effect of backwards propagation of bounds. We denote the directions of bound propagation by forward bound propagation (FBP) and backward bound propagation (BBP), respectively.

To simplify the notation, we assume in the following that D and E are hyperrectangles included in $B^0 = [L^0, U^0]$ and $B^K = [L^K, U^K]$.

4.1. Relaxations of ReLU networks

We define the feasible set of the LP relaxation of (4) as

$$R_R(\Theta, B) := \{(x, s, z) : \text{(4a)}, \text{(4b)}, \text{(4d)} - \text{(4f) are satisfied}, 0 \leq z \leq 1\}, \tag{8}$$

where the binary restrictions on z are replaced by $0 \leq z \leq 1$. This relaxation can be interpreted as a relaxation of the ReLU logic, and we refer to it as *ReLU relaxation*. The ReLU relaxation makes it possible to increase the output from the ReLU and we get the following bounds on the output from node j of layer k:

$$(W^k x^{k-1} + b^k)_j \leq x_j^k \leq U_j^k \frac{(W^k x^{k-1} + b^k)_j - L_j^k}{U_j^k - L_j^k} \tag{9}$$

We note two things from (9). First, even if $(W^k x^{k-1} + b^k)_j < 0$ the unit can still give a positive output in the ReLU relaxation. Second, the upper bound depends on both U_j^k and L_j^k, and a decrease in U_j^k or increase in L_j^k directly lowers the bound. This clearly motivates a study of BT procedures.

We use the notation $R_R(\Theta, B, I)$ to denote a partial ReLU relaxation for which the ReLUs of nodes (j, k) in the index set I are relaxed. A MILP problem results from a partial ReLU relaxation. We let the LP relaxation $R_R(\Theta, B)$ correspond to $R_R(\Theta, B, I)$ with all nodes indexed by I.

When considering the bound tightening of a node (j, k), it is possible to relax $P(\Theta, B)$ by removing constraints related to other layers than k from (4). We may for example remove all layers following k $(k+1, \ldots, K)$ to simplify the problem while retaining FBP from layer 0 through k. A weaker relaxation can be obtained by removing all layers but k and $k-1$. This relaxation can be used to propagate bounds forward one layer at the time. In both these cases, where all layers following k are excluded, node (j, k) can be considered an output node. We may then also remove the constraints of all the other nodes in layer k to reduce the problem size. We name this type of relaxation *layer relaxation* and denote it by $R_L(\Theta, B, I)$, where we specify by the index set I which nodes to remove/relax.

For the relaxations discussed above, we have that $P(\Theta, B) \subseteq R_R(\Theta, B, I)$ and $P(\Theta, B) \subseteq R_L(\Theta, B, I)$ for any index set I. Thus, a BT procedure utilizing these relaxation produces *valid* bounds; i.e., the tightened bounds do not diminish the set of input-output pairs $F(\Theta, B)$.

4.2. Prototype for bound tightening procedures

The bound tightening procedures we discuss subsequently are special cases of the prototype in Algorithm 1.

Given a network parameterized by Θ, some initial bounds B^- and a scheme S, the prototypical procedure computes tightened bounds B. The scheme S state for each node (j, k) which relaxations that should be used. Bounds l_j^k and u_j^k of each node (j, k) are computed so that $l_j^k \leq x_j^k - s_j^k \leq u_j^k$. The tightness of these bounds is decided by the constraint set C_j^k, representing a subset of the constraints in

Algorithm 1 Prototype for bound tightening procedures

```
Require: Parameters $\Theta$, initial bounds $B^-$ and scheme $S$
  $B \leftarrow B^-$
  $k \leftarrow 0$
  for $k \leq K$ do                                                    ▷ Iterate over layers
      $j \leftarrow 1$
      for $j \leq n_k$ do                                              ▷ Iterate over nodes
          Build constraint set $C_j^k$ from $\Theta$ and $B$ following $S$
          Solve for upper bound $u_j^k = \max x_j^k - s_j^k$ s.t. $C_j^k$
          Solve for lower bound $l_j^k = \min x_j^k - s_j^k$ s.t. $C_j^k$
          Update bounds $B$: $U_j^k \leftarrow u_j^k$ and $L_j^k \leftarrow l_j^k$
          $j \leftarrow j + 1$
      end for
      $k \leftarrow k + 1$
  end for
  return Tightened bounds $B$
```

(4), or a valid relaxation of these. The bounds on x_j^k and s_j^k are then updated using the rules in (5). We note that, for the last (linear) layer K, the objective function of the upper and lower bound problems simplifies to x_j^K.

If a strictly positive lower bound is found for x_j^k, i.e. $L_j^k > 0$, the ReLU may be relaxed by fixing $z_j^k = 1$ and $s_j^k = 0$ (or removing them from the problem). Similarly, if $U_j^k < 0$, we may fix $z_j^k = 0$ and $x_j^k = 0$; for this case we say that the neuron (j, k) is dead. This effectively reduces the number of binary variables in the formulation.

When tightening the bounds of (4), we should consider in which order to processes the nodes. For FBP, it is natural to process nodes in the order of layers, beginning with the nodes in the first hidden layer ($k = 1$). However, when bound information can flow backwards from the output bounds B^K, it is not obvious in which order the nodes should be processed. In Appendix A, we provide an argument for using the same strategy also when bound information may propagate backwards.[2] The argument is that, since BBP is much weaker than FBP, the bound tightening procedure should focus on propagating bounds forward. In line with this argument, Algorithm 1 makes one forward pass through the layers, processing the nodes of each layer in the order they are indexed. We may apply the argument again when considering the ordering within each layer. Since the only interaction between the nodes in a layer is via BBP on subsequent nodes, different processing orders will likely perform similarly.

We note that k is initialized to 0, since the input bounds $[L^0, U^0]$ may be tightened by the output bounds when the constraint sets C_j^k allow for BBP. For instances of the procedure for which BBP cannot occur due to the choice of constraint set C_j^k, k may be initialized to 1.

4.3. Feasibility-based bound tightening (FBBT)

FBBT subsumes BT procedures that use purely primal feasibility arguments and remove parts of the domains in which no feasible solutions are contained [30]. These procedures rely on interval arithmetic to compute the bounds on constraint activations over the variable domains, and conversely, propagating the bounds on the constraint activities back to the variable domains.

Using simple interval arithmetic we may infer bounds $l_j^k \leq x_j^k - s_j^k \leq u_j^k$ for node (j, k) in layer $k \geq 2$ as follows:

$$\begin{aligned} u_j^k &= \sum_{i=1}^{n_{k-1}} \max\left\{w_{ji}^k \max\{0, U_i^{k-1}\}, w_{ji}^k \max\{0, L_i^{k-1}\}\right\} + b_j^k, \\ l_j^k &= \sum_{i=1}^{n_{k-1}} \min\left\{w_{ji}^k \max\{0, U_i^{k-1}\}, w_{ji}^k \max\{0, L_i^{k-1}\}\right\} + b_j^k, \end{aligned} \tag{10}$$

where we have utilized the bounds on x_j^{k-1} in (5). The same bounds can be found by solving the LP

[2]These procedures must not be confused with the forward and backward propagation used in inference and training of neural networks.

problems:

$$\begin{aligned} u_j^k &= \max\left\{y : y \in C_j^k\right\}, \\ l_j^k &= \min\left\{y : y \in C_j^k\right\}, \end{aligned} \tag{11}$$

for the constraint set

$$C_j^k = \left\{y : y = w_j^k x + b^k, x \in [\max\{0, L^{k-1}\}, \max\{0, U^{k-1}\}] \subset \mathbb{R}^{n_{k-1}}\right\}. \tag{12}$$

To compute the bounds in the first hidden layer $k = 1$, we remove the inner max-operators in (10) or (12), since there is no ReLU operator on the input layer.

For a DNN, we may propagate the input bounds forward by solving (10) or (11), and then update the bounds according to (5) for units in successive layers, beginning with the units in layer $k = 1$. That is, we follow the prototypical procedure in Algorithm 1, but initialize with $k = 1$. This procedure employs both layer and ReLU relaxations, as seen from C_j^k, and we refer to it as LRR in the rest of this paper. The procedure is computationally cheap since it considers only the constraints in (4b) for the node being tightened.

In general, feasibility-based procedures compute bounds that are sub-optimal, due to the weak relaxations. Stronger BT procedures can be devised by including more constraints and exploiting integer information. For example, the LRR procedure propagates bounds in the forward direction only, and may not take advantage of the output bounds B^K. To do so, a BT procedure must be able to perform BBP, either by starting at the last layer, or by including enough constraints to link the variable domain being tightened to the output bounds. Next, we consider some alternative BT procedures that use stronger relaxations, but are more computationally demanding.

4.4. Optimization-based bound tightening (OBBT)

We devise several OBBT procedures for ReLU networks using the relaxations in Section 4.1. These procedures rely on solving a series of $2\sum_{i=0}^{K} n_k$ optimization problems to find the tightest variable bounds on relaxations of (4).

RR procedure. First, we consider an OBBT procedure using the LP relaxation $R_R(\Theta, B)$ of (4). The procedure computes bounds on $x_j^k - s_j^k$ by solving the optimization problems:

$$\begin{aligned} u_j^k &= \max\left\{x_i^k - s_i^k : (x, s, z) \in R_R(\Theta, B)\right\}, \\ l_j^k &= \min\left\{x_i^k - s_i^k : (x, s, z) \in R_R(\Theta, B)\right\}, \end{aligned}$$

for $k = 0, \dots, K$ and $j = 1, \dots, n_k$. Setting $C_j^k = R_R(\Theta, B)$ in Algorithm 1, we obtain an LP-based OBBT procedure which we call RR.

ITER-RR procedure. With RR, the input bounds B^0 are propagated forward. The relaxation also links all nodes to the output bounds B^K, but we do not expect this information to propagate far (in the backwards direction) due to the looseness of the LP relaxation. To improve the quality of the bounds, we may run the algorithm iteratively until no significant reduction in the bounds is observed or until a maximum number of iterations is reached. We measure the improvement in the bounds from one iteration to the next using the mean absolute distance (MAD) statistic in Appendix B. The iterative RR procedure (ITER-RR) is given in Algorithm 2, Appendix C.

The procedures RR and ITER-RR are both pure LP procedures. The subsequent procedures require solving $2\sum_{i=0}^{K} n_k$ MILP problems of increasing complexity (the number of constraints and binary variables increase with layer depth).

LR procedure. Next, we consider a procedure that utilizes layer relaxations $R_L(\Theta, B, I_j^k)$. When solving for node (j, k), we construct the index set I_j^k so that the constraints are removed for all nodes except (j, k) and nodes in preceding layers $\{0, \dots, k-1\}$. In the framework of Algorithm 1, we initialize from $k = 1$ and set $C_j^k = R_L(\Theta, B, I_j^k)$. We name this procedure LR since it only utilizes layer relaxations.

SEMI-RR procedure. The LR procedure removes all layers following a node (j, k) and may not propagate bounds backwards. To address this potential weakness, we devise a procedure that keeps all following layers, but relaxes their ReLU operators. The procedure is given by Algorithm 1 with $C_j^k = R_R(\Theta, B, I_j^k)$, where I_j^k indexes the nodes in layers $\{k, \dots, K-1\}$ (excluding layer K since it is affine). The procedure is named SEMI-RR since it relaxes the ReLUs in layer k onwards.

NO-R procedure. Finally, we may compute the tightest possible bounds of a ReLU network by using the full constraint set in (4); i.e. we set $C_j^k = P(\Theta, B)$. The procedure does not relax any constraints and we therefore name it NO-R.

4.5. Summary of bound tightening procedures

The various BT procedures discussed above employ two different types of relaxations. They either relax the integrality constraint on the binary variables z (ReLU relaxation), or remove nodes or layers and their respective constraints (layer relaxation), or utilize both types of relaxations. The procedures, which we have named based on the relaxations they employ, are listed in Table 2.

Table 2: Bound tightening procedures.

BT procedure	BT type	Subproblem class	Layer relaxation	ReLU relaxation	Backward propagation**
LRR	FBBT	LP/interval arithmetic	✓	✓	–
RR	OBBT	LP	–	✓	✓
ITER-RR	OBBT	LP	–	✓	✓
LR	OBBT	MILP	✓	-	–
SEMI-RR	OBBT	MILP	–	*	✓
NO-R	OBBT	MILP	–	–	✓

*ReLUs of preceding nodes are not relaxed.
**BT procedure can propagate output bounds backwards through the network.

Let B^{LRR} be the bounds found by running LRR, B^{RR} by RR, and so on. By inspecting Table 2 we see that $\text{MAD}(B^{\text{ITER-RR}}) \leq \text{MAD}(B^{\text{RR}}) \leq \text{MAD}(B^{\text{LRR}})$. Likewise, we have that $\text{MAD}(B^{\text{NO-R}}) \leq \text{MAD}(B^{\text{SEMI-RR}}) \leq \text{MAD}(B^{\text{LR}})$, given that subproblems are not time limited. In the absence of output bounds, we have that $\text{MAD}(B^{\text{NO-R}}) = \text{MAD}(B^{\text{SEMI-RR}}) = \text{MAD}(B^{\text{LR}})$, and LR is clearly the cheaper procedure. It is not trivial to compare the LP-based procedures with MILP-based procedures since they employ different relaxations. However, we do have that $\text{MAD}(B^{\text{SEMI-RR}}) \leq \text{MAD}(B^{\text{RR}})$, again assuming that subproblems are not time limited.

Several of the presented BT procedures have been used in other works. The LRR was employed by [15], while [18] used the LR procedure.

Remark 1 (Pre-computing bounds). Due to their computational cost, we only invoke the BT procedures once before optimization (or at the root node of the branch-and-bound tree). The bounds can then be stored and reused in subsequent optimizations. We rely on the solver's bound tightening capabilities further down the branch-and-bound tree.

Remark 2 (Subproblem time limit). For the MILP-based procedures LR, SEMI-RR, and NO-R we may limit the solution time of each subproblem to reduce the overall computational cost. For subproblems that do not terminate within this time limit we use the best bound found by the solver, ensuring that the computed bounds are still valid. We introduce a naming convention where we postfix the BT procedure name by the subproblem solution time limit (in seconds). For example, we write *LR(60)* for the LR procedure with the solution time of the subproblems limited to 60 seconds.

Remark 3 (Bound initialization). We initialize all OBBT procedures with bounds computed by LRR. This initialization is cheap and gives the procedures an identical starting point. Furthermore, it allows us to compare the procedures with the mean relative distance (MRD) statistic in Appendix B.

5. Numerical Study

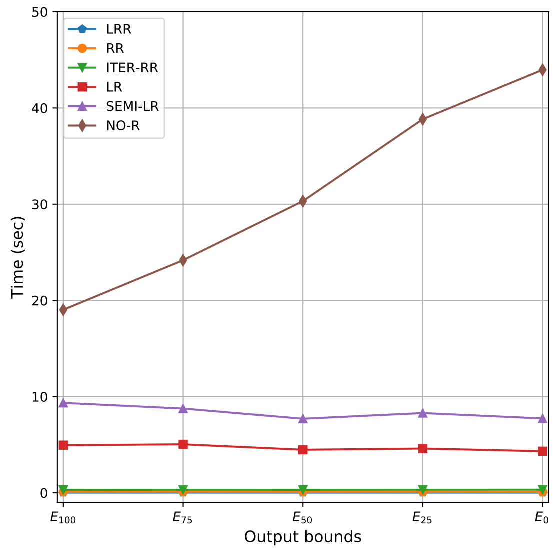
We provide a computational study to evaluate the feasibility of using ReLU networks as surrogate models in mixed-integer programs. We focus primarily on solution times for different network architectures and bound tightening procedures in our investigation. The BT procedures in Section 4 are compared based on computational efficiency and the statistics in Appendix B, as well as optimization solution time.

We first evaluate the BT procedures' ability to propagate output bounds backwards on a set of randomly generated neural networks. Next, we solve a series of increasingly challenging optimization problems to test the practical performance of the big-M formulation in (4) with bounds computed by the proposed BT procedures. We finally solve an oil production optimization problem including ten ReLU network surrogate models.

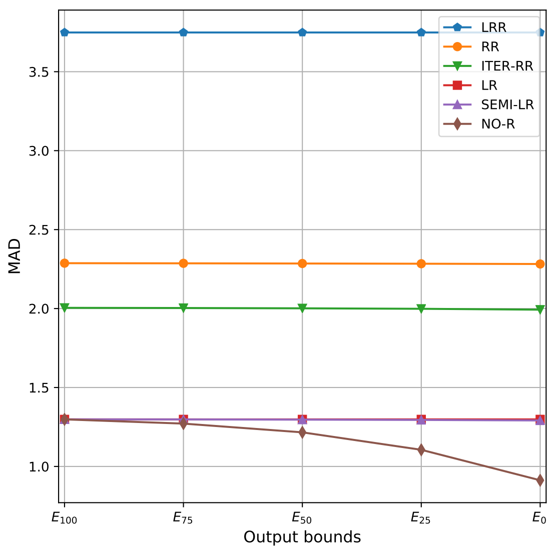
The neural networks are initialized and trained using TensorFlow [21]. We solve all optimization problems using Gurobi 8.1 with default settings [31], on a machine equipped with an Intel Core i7-8700K processor and 32 GB of RAM memory.

5.1. Bound tightening for randomly initialized ReLU networks with output bounds

We generate ten networks with layers $(3, 20, 20, 10, 1)$ to investigate the BT procedures' ability to propagate output bounds backwards. The networks parameters are initialized using the method by [32], so that for inputs $x^0 \sim \mathcal{N}(0,1)$, the output $x^K \sim \mathcal{N}(0,1)$. We set the input bounds to $B^0 = [-1,1]^3$ and output bounds to $B^K = E_{100} := [-1,1]$. We then proceed by running the procedures for diminishing output bounds: $E_{75} = [-0.75, 0.75]$, $E_{50} = [-0.5, 0.5]$, $E_{25} = [-0.25, 0.25]$, and $E_0 = [0,0] = \{0\}$. The results are shown in Figure 3, where time and MAD values are averaged over the ten generated networks.



(a)



(b)

Figure 3: Average solution time (a) and MAD (b) for ten neural networks are shown for various BT procedures. NO-R is the only procedure that significantly reduces the MAD as the output bounds are diminished.

Comparing the MAD values in Figure 3b for E_{100}, we see that LRR produces the loosest bounds, as expected. Next follows RR and ITER-RR, with the latter yielding a lower MAD on average. The MILP-based BT procedures, LR, SEMI-LR, and NO-R, obtain the same average MAD value. This is due to the fact that the output bounds E_{100} have no tightening effect on preceding units; that is, $f(x) \in E_{100}$ for all $x \in B^0$. However, as the output bounds are tightened, we see that the average MAD decreases for NO-R. For comparison, we report the average MAD values for E_0 and E_{100} (denoted MAD_0 and MAD_{100}) in Table 3. As reported in the table, NO-R is the only BT procedure capable of significantly reducing the MAD by propagating the E_0 bounds backwards.

Table 3: Average MAD for output bounds E_0 and E_{100}.

BT procedure	MAD_{100}	MAD_0	$100 \times \text{MAD}_0/\text{MAD}_{100}$
LRR	3.65616	3.65616	100.00
RR	2.24987	2.24862	99.94
ITER-RR	1.94224	1.91441	98.57
LR	1.25684	1.25684	100.00
SEMI-LR	1.25684	1.24049	98.70
NO-R	1.25684	0.95079	**75.65**

Turning to Figure 3a, we see that the average solution times of the BT procedures are ordered as expected. The solution time increases with the number of constraints included in the BT procedure, and jumps considerably as binary variables are included. The ordering is the same as in the legend of the figure, LRR having the lowest and NO-R the highest solution times on average. The most interesting observation in this figure is that the solution times for NO-R increases as the output bound is tightened, while the other BT procedures seem to be unaffected. Possible explanations are that NO-R, to an increasing degree, must propagate the output bounds backwards, thus linking more variables (nodes) in the MILP problems and that finding feasible integer solutions are harder with a tight output bound. Furthermore, the constraints through which the output bounds must be propagated backwards are likely quite weak (see discussion in Appendix A), which may affect the numerical performance of the MILP solver.

5.2. Optimization of approximated n-dimensional quadratic functions

We consider n-dimensional quadratic functions

$$q(x) = x^\top Ax + b^\top x + c,$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, and $c \in \mathbb{R}$. The coefficients are drawn as $A_{ij} \sim \mathcal{N}(0, 5)$, $b_i \sim \mathcal{N}(0, 1)$, and $c \sim \mathcal{N}(0, 1)$. The quadratic functions resulting from this construction are likely to be indefinite and non-separable, and thus difficult to optimize. Furthermore, the higher variance on the coefficients of the quadratic terms increases the curvature of the generated functions. Due to the curvature, many pieces are required to obtain an accurate piecewise-linear approximation of these functions.

For $n = 1, \ldots, 6$, we generate ten quadratic functions and approximate them by ReLU networks with the configurations given in Table 4. The table also lists the number of training samples, and the resulting *mean absolute percentage error* (MAPE) averaged over the ten models. One of the two-dimensional approximations is shown in Figure 4. For the displayed case, the drawn A matrix is indefinite, leading to a saddle surface.

Table 4: ReLU network approximations of quadratic functions.

n	Layers	# parameters	# training samples	MAPE (%)
1	[1, 10, 5, 1]	81	100	2.3
2	[2, 20, 10, 1]	281	500	3.5
3	[3, 40, 20, 1]	1 001	2 000	3.7
4	[4, 50, 20, 1]	1 291	5 000	3.7
5	[5, 50, 30, 30, 1]	2 791	10 000	2.6
6	[6, 80, 40, 40, 1]	5 481	20 000	2.0

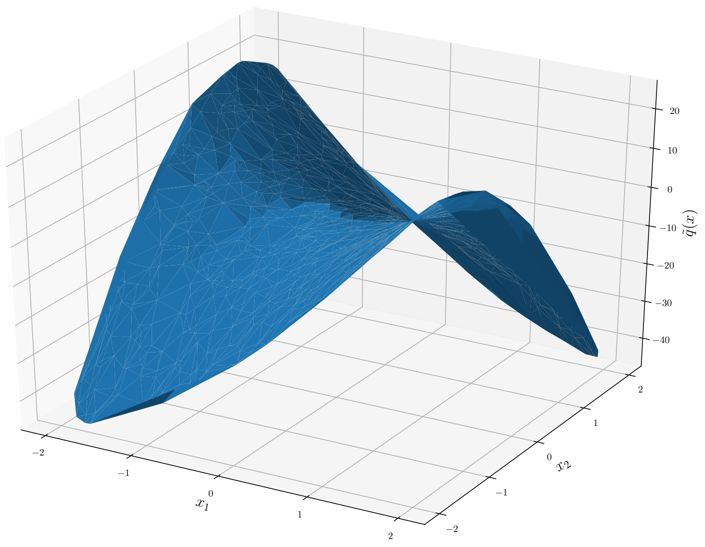


Figure 4: ReLU network approximation of a quadratic function with an indefinite A matrix.

Using the ReLU approximations, we consider optimization problems on the following form:

$$\min \{ f_{\Theta_1}(x) : f_{\Theta_2}(x) = \alpha, x \in [-1, 1]^n \}, \qquad (Q_n)$$

where f_{Θ_1} and f_{Θ_2} are ReLU networks, and α is a real constant. For $n = 1, \dots, 6$, we construct five optimization problems by pairing the ReLU networks described above.

We set the objective function to f_{Θ_1}, and optimize under the constraint that the second network, f_{Θ_2}, must intersect a plane at level α at the solution. Notice that the feasible set, specified by the constant $\alpha \in \mathbb{R}$, is disconnected when f_{Θ_2} has multiple crossings of the α-plane. This can be seen for the saddle surface in Figure 4, for which the intersection to an α-plane is a hyperbola (unless the α-plane intersects the saddle point).

The optimization results are given in Table 5. When no feasible solution is found within the total time for one or more of the networks, we mark this with ∞ in the *Gap* column. From the table, we see that the LRR procedure has the best performance for the low dimensional problems ($n = 1, 2, 3$). A likely explanation is that the solver's internal BT procedures work well for small network sizes, and that the stronger bound tightening procedures lead to computational overhead. For $n = 4$, the RR procedure takes the lead by benefiting from tighter bounds without spending to much time in the BT procedure. For the largest problems, $n = 5$ and $n = 6$, the LR(1) procedure is the best performer in terms of total solution time. The trend seems to be that larger network sizes benefit more from stronger bound tightening. We note that for $n = 6$, we are able to solve all five instances when using the LR(1) procedure. Even with subproblem solution time is limited to one second, the SEMI-LR(1) and NO-R(1) procedures use several hundred seconds on bound tightening, leaving little time for solving the optimization problems within the total time limit of 600 seconds.

Table 5: Average solution times and optimality gap for Q_n (five instances for each value of n). Solution times for bound tightening (T_{BT}) and optimization (T_{OPT}) are given in seconds. Total solution time, $T_{\mathrm{BT}} + T_{\mathrm{OPT}}$, is limited to 600 seconds.

n	BT procedure	T_{BT}	T_{OPT}	$T_{\mathrm{BT}} + T_{\mathrm{OPT}}$	Gap (%)	# solved
1	LRR	0.013	0.003	**0.016**	0	5/5
	RR	0.024	0.002	0.026	0	5/5
	ITER-RR	0.049	0.002	0.051	0	5/5
	LR(1)	0.054	0.003	0.057	0	5/5
	SEMI-LR(1)	0.138	0.002	0.140	0	5/5
	NO-R(1)	0.070	0.001	0.071	0	5/5
2	LRR	0.024	0.070	**0.094**	0	5/5
	RR	0.072	0.056	0.128	0	5/5
	ITER-RR	0.232	0.048	0.280	0	5/5
	LR(1)	0.374	0.042	0.416	0	5/5
	SEMI-LR(1)	1.575	0.047	1.622	0	5/5
	NO-R(1)	1.959	0.037	1.996	0	5/5
3	LRR	0.065	0.418	**0.48**	0	5/5
	RR	0.238	0.274	0.51	0	5/5
	ITER-RR	0.793	0.243	1.04	0	5/5
	LR(1)	3.772	0.237	4.01	0	5/5
	SEMI-LR(1)	17.42	0.235	17.66	0	5/5
	NO-R(1)	36.65	0.206	36.86	0	5/5
4	LRR	0.082	3.497	3.58	0	5/5
	RR	0.315	1.414	**1.73**	0	5/5
	ITER-RR	0.841	1.375	2.22	0	5/5
	LR(1)	7.717	1.217	8.93	0	5/5
	SEMI-LR(1)	32.02	1.273	33.29	0	5/5
	NO-R(1)	84.29	0.528	84.82	0	5/5
5	LRR	0.164	204.1	204.3	10.6	4/5
	RR	1.174	142.4	143.6	5.9	4/5
	ITER-RR	5.245	135.9	141.1	4.4	4/5
	LR(1)	95.59	13.2	**108.8**	0	5/5
	SEMI-LR(1)	181.2	17.7	198.9	0	5/5
	NO-R(1)	303.9	16.0	319.9	0	5/5
6	LRR	0.286	433.0	433.3	∞	2/5
	RR	3.174	389.5	392.7	7.9	2/5
	ITER-RR	14.44	292.4	306.8	2.5	3/5
	LR(1)	195.8	78.1	**273.9**	0	5/5
	SEMI-LR(1)	335.0	128.2	463.2	∞	3/5
	NO-R(1)	472.9	69.0	541.9	2.7	3/5

The results given in Table D.10, Appendix D, show that the BT procedures that utilize stronger relaxations also compute tighter bounds, as expected. Another expected observation is that the number of dead neurons identified is strongly correlated with bound tightness.

In terms of BT solution times, we see that the interval arithmetic-based LRR procedure scales well with network size. The LP-based procedures scale moderately, while the MILP-based procedures scale poorly and quickly become computationally demanding. For the larger networks, the subproblem time limit of one second helps to limit BT solution times, at the cost of looser bounds. This explains why LR, which has the smallest subproblems among the MILP-based procedures, computes the tightest bounds for $n = 6$.

5.3. Oil production optimization case

To test the practical performance of the proposed methods we solve a production optimization case from an offshore petroleum production field involving eight subsea wells producing oil, gas, and water. Each of the wells produce to a topside processing facility, a platform, via one of two risers (pipelines transporting the fluid from the seabed to the topside processing facility).

An offshore production system consists of several interconnected modules with corresponding interdependencies. The reservoir is the subsurface structure where oil and gas are located prior to extraction. A number of wells are drilled into the reservoir. The flow coming from a well is a mixture of water, oil and gas, called production phases, and is routed to a platform through a network of pipelines. Manifolds are used to connect different pipelines and to mix the incoming flows. At the platform, separators split the production phases and route the resulting streams of oil and gas to an export line. The export lines lead production phases off-site. The production flow from a well may be routed to a subset of the separators on the platform. A well can only be routed to a single separator at any given time. The routing decision must take into consideration the interdependencies between wells which produce to the same separator.

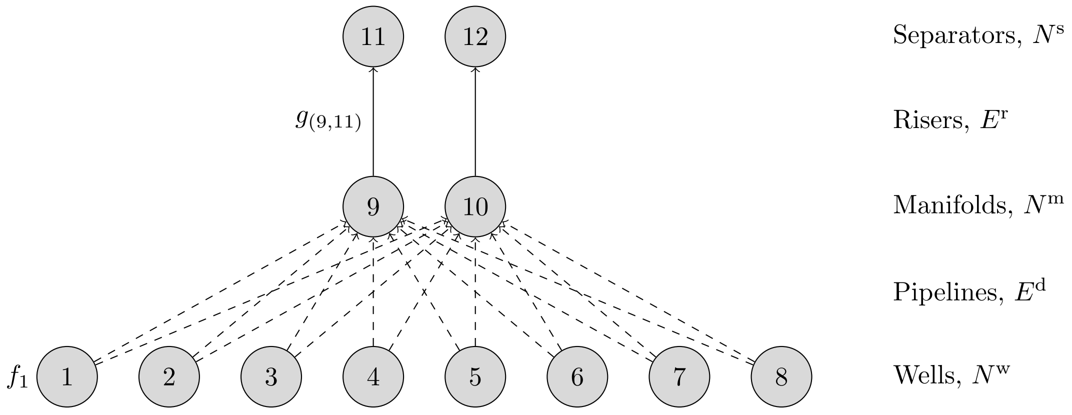

11
12
Separators, N^{s}
$g_{(9,11)}$
Risers, E^{r}
9
10
Manifolds, N^{m}
Pipelines, E^{d}
f_1
1
2
3
4
5
6
7
8
Wells, N^{w}

Figure 5: Production system flow graph, with nodes represented by grey circles and edges by arrows. Discrete edges are dashed. The nonlinearities f_1 and $g_{(9,11)}$, related to Node 1 and Edge (9,11), are shown.

The production system is modeled by a directed acyclic graph $G = (N, E)$, with nodes N and edges E. An illustration of the graph is given in Figure 5. To simplify modeling, we use the utility sets in Table 6. The set of nodes is $N = N^{\mathrm{w}} \cup N^{\mathrm{m}} \cup N^{\mathrm{s}}$, and the set of edges is $E = E^{\mathrm{d}} \cup E^{\mathrm{r}}$. As illustrated in Figure 5, each well node ($i \in N^{\mathrm{w}}$) has two leaving *discrete* edges. By allowing zero or one of these edges to be active, we model routing and on/off switching of well flows.

Table 6: Utility sets

Set	Description
N	Set of nodes in the network.
N^{w}	Set of *well* (source) nodes in the network. $N^{\mathrm{w}} \subset N$.
N^{m}	Set of *manifold* nodes in the network. $N^{\mathrm{m}} \subset N$.
N^{s}	Set of *separator* (sink) nodes in the network. $N^{\mathrm{s}} \subset N$.
E	Set of edges in the network. An edge $e = (i, j)$ connects node i to node j, where $i, j \in N$.
E^{d}	Set of *discrete* edges that can be *open* or *closed*. $E^{\mathrm{d}} \subset E$.
E^{r}	Set of *riser* edges. $E^{\mathrm{r}} \subset E$.
E_i^{in}	Set of edges entering node i, i.e. $E_i^{\mathrm{in}} = \{e : e = (j, i) \in E\}$.
E_i^{out}	Set of edges leaving node i, i.e. $E_i^{\mathrm{out}} = \{e : e = (i, j) \in E\}$.
C	$C = \{\mathrm{oil}, \mathrm{gas}, \mathrm{wat}\}$, denoting the flow rate of oil, gas, and water, respectively.

The variables are the pressure p_i at each node $i \in N$, the pressure drop Δp_e on each edge $e \in E$, the flow rate $q_{e,c}$ of phase $c \in C$ on edge $e \in E$, and the binary variable y_e on the discrete edge $e \in E^{\mathrm{d}}$. The variables y_e are used to model routing and on/off switching of wells. The problem also includes constants $c_{e,\mathrm{gor}}$ (gas-oil ratio), $c_{e,\mathrm{wor}}$ (water-oil ratio), flow rate bounds $q^L_{e,c}$ and $q^U_{e,c}$, pressure bounds p^L_i and p^U_i, and separator pressures p^s_i, which are set to realistic values. For further details about the modeling approach we refer the reader to [7].

The complete formulation of the production optimization problem is given below. The objective is to maximize oil production by routing and choking the well flows, while honoring constraints on momentum balances and variable bounds.

$$
\begin{aligned}
\underset{y,q,p,\Delta p}{\text{maximize}} \quad & z = \sum_{e \in E^{\mathrm{r}}} q_{e,\mathrm{oil}} && \\
\text{subject to} \quad & \sum_{e \in E^{\mathrm{in}}_i} q_{e,c} = \sum_{e \in E^{\mathrm{out}}_i} q_{e,c}, && \forall c \in C, i \in N^{\mathrm{m}} \\
& p_j = g_e(q_{e,\mathrm{oil}}, q_{e,\mathrm{gas}}, q_{e,\mathrm{wat}}, p_i), && \forall e \in E^{\mathrm{r}} \\
& \Delta p_e = p_i - p_j, && \forall e \in E^{\mathrm{r}} \\
& -M_e(1-y_e) \le p_i - p_j - \Delta p_e, && \forall e \in E^{\mathrm{d}} \\
& p_i - p_j - \Delta p_e \le M_e(1-y_e), && \forall e \in E^{\mathrm{d}} \\
& \sum_{e \in E^{\mathrm{out}}_i} y_e \le 1, && \forall i \in N^{\mathrm{w}} \\
& y_e q^L_{e,c} \le q_{e,c} \le y_e q^U_{e,c}, && \forall c \in C, e \in E^{\mathrm{d}} \\
& p^L_i \le p_i \le p^U_i, && \forall i \in N \\
& \sum_{e \in E^{\mathrm{out}}_i} q_{e,\mathrm{oil}} = f_i(p_i), && \forall i \in N^{\mathrm{w}} \\
& \sum_{e \in E^{\mathrm{out}}_i} q_{e,\mathrm{gas}} = c_{e,\mathrm{gor}} \sum_{e \in E^{\mathrm{out}}_i} q_{e,\mathrm{oil}}, && \forall i \in N^{\mathrm{w}} \\
& \sum_{e \in E^{\mathrm{out}}_i} q_{e,\mathrm{wat}} = c_{e,\mathrm{wor}} \sum_{e \in E^{\mathrm{out}}_i} q_{e,\mathrm{oil}}, && \forall i \in N^{\mathrm{w}} \\
& p_i = p^s_i = \text{const.}, && \forall i \in N^{\mathrm{s}} \\
& y_e \in \{0,1\}, && \forall e \in E^{\mathrm{d}}
\end{aligned}
\tag{PO}
$$

Problem *PO* contains a total of ten nonlinearities that we model with ReLU networks: eight well performance curves $f_i(p_i)$ for $i \in N^{\mathrm{w}}$, and two riser pressure drop functions $g_e(q_{e,\mathrm{liq}}, q_{e,\mathrm{gas}}, p_i)$ for $e \in E^{\mathrm{r}}$. A well performance curve $f_i(p_i)$ for $i \in N^{\mathrm{w}}$ relates the pressure at well i, p_i, with the rate of flow out of the node $q_{(e,oil)}$ for $e \in E^{\mathrm{out}}_i$. The riser pressure drop functions $g_e(q_{e,\mathrm{liq}}, q_{e,\mathrm{gas}}, p_i)$ for $e \in E^{\mathrm{r}}$ calculates the pressure at the separator as a function of the flow rates $q_{e,\mathrm{liq}}, q_{e,\mathrm{gas}}$ in the riser and the pressure at the start of the riser p_i. To test the effect of network depth on solution time, we consider the shallow and deep network architectures given in Table 7. The networks are trained on scattered data sampled from a multiphase flow simulator. Since the simulated data is free of noise, the DNNs are trained with a low L_2 penalty on the parameters. The ReLU networks achieve a mean absolute percentage error (MAPE) of less than 1%, and approximate the simulator with high accuracy.

Table 7: Deep neural networks for wells and risers.

DNN	Layers	# weights	# 0-1 var.	MAPE (%)
Shallow well nets	(1, 20, 20, 1)	481	40	0.73
Shallow riser nets	(4, 50, 50, 1)	2851	100	0.30
Deep well nets	(1, 10, 10, 10, 10, 1)	361	40	0.48
Deep riser nets	(4, 20, 20, 20, 20, 20, 1)	1801	100	0.42

The MILP formulations of the shallow and deep networks have the same number of binary variables: 40 for the wells and 100 for the risers. The resulting optimization problem *PO* has a total of 536 binary variables, including 16 binary variables for well routing.

The BT procedures achieve the values reported in Tables D.11-D.12 for the well networks and Tables D.13-D.14 for the riser networks (see Appendix D). Due to the constraints on the separator pressures, the riser networks are output bounded to the singleton $\{p^s_i\}$ for $i \in N^{\mathrm{s}}$. Looking at the MRD values for the riser networks, it is evident that NO-R computes tighter bounds than the other BT procedures which are unable to fully utilize the output bounds via BBP. As can be seen from the solution times, the NO-R procedure is computationally expensive. For the shallow riser networks the average run time is 306 seconds, while for the deep riser networks it is 5080 seconds. Limiting the subproblem solution time to 60 seconds helps for the deeper networks, bringing the average solution time down to 1771 seconds for NO-R(60). The limitation in solution time do not seem to affect the bound tightness by much for these networks sizes, and NO-R(60) achieves a MRD of 0.1 % compared to NO-R. For the well networks and the shallow riser networks, the subproblem time limit is never effective, which means that NO-R and NO-R(60) compute equivalently.

With the tightened bounds, we solve the production optimization problem (PO). The results are given in Tables 8 and 9 for the shallow and deep ReLU networks, respectively. The effect of bound tightness on optimization solution time is striking. With the shallow networks, we are only able to close the optimality gap within one hour using the bounds computed using the LR(60), NO-R(60) and NO-R procedure. For the deep networks, the gap is closed when we use bounds computed by the NO-R(60) and NO-R procedure. Notice that the comparison is somewhat unfair since the overall computation time used by the NO-R procedure far exceeds that of the other BT procedures. It is, however, interesting to compare the optimization solution times (T_{OPT}) for the various BT procedures.

With NO-R(60), we solve both the deep and shallow problem within one hour, which is acceptable for a practical application PO. With the optimal bounds computed by NO-R for the deep networks, the optimization converges in just 5 seconds. The expensive BT procedures thus show some merit on these problems, especially for applications where it is interesting to solve the problem many times; perhaps with small adjustments to the problem for each optimization run (required that the adjustments do not invalidate the computed bounds).

Table 8: Solution times for production optimization problem with shallow ReLU networks. Bound tightening (T_{BT}) and optimization (T_{OPT}) solution times are given in seconds. T_{OPT} is limited to 3600 seconds. $z^\star$ is the objective value of the best found solution and Gap is the difference between the upper and lower bounds relative to the lower bound.

BT procedure	T_{BT}	T_{OPT}	$T_{\mathrm{BT}} + T_{\mathrm{OPT}}$	$z^\star$	Gap (%)
LRR	0.3	3600	3600	1.2864	7.5
RR	1.3	3600	3601	1.2864	4.7
ITER-RR	2.5	3600	3603	1.2864	4.7
LR(60)	18.6	2383	2402	1.2864	0
SEMI-RR(60)	115.0	3600	3715	1.2864	15.5
NO-R(60)	613.2	20	**633**	1.2864	0
NO-R	613.2	20	**633**	1.2864	0

Table 9: Solution times for production optimization problem with deep ReLU networks. Bound tightening (T_{BT}) and optimization (T_{OPT}) solution times are given in seconds. T_{OPT} is limited to 3600 seconds. $z^\star$ is the objective value of the best found solution and Gap is the difference between the upper and lower bounds relative to the lower bound.

BT procedure	T_{BT}	T_{OPT}	$T_{\mathrm{BT}} + T_{\mathrm{OPT}}$	$z^\star$	Gap (%)
LRR	0.3	3600	3600	–	–
RR	1.4	3600	3601	–	–
ITER-RR	11.4	3600	3611	–	–
LR(60)	220.1	3600	3820	–	–
SEMI-RR(60)	273.8	3600	3874	–	–
NO-R(60)	3543.2	57	**3600**	1.3049	0
NO-R	10161.2	5	10166	1.3049	0

Remark 4. As an alternative approach to ReLU networks, we may model the well and riser nonlinearities using a traditional MILP formulation that interpolates the sample points, as discussed in Section 2. For each nonlinearity, we use Delaunay triangulation on the samples to partition the domain into simplices. With the partitions, we obtain PWL approximations using the DLog model in [8]. The resulting formulation of PO has 90 binary variables, 718 656 continuous variables, and 287 constraints. When attempting to solve this problem, the solver quickly runs out of memory without finding any feasible

solution. We thus deem this approach unsuccessful for the production optimization case with irregularly sampled data points.

6. Concluding remarks

As we discussed in Section 2, ReLU networks offer practitioners a versatile framework for modeling PWL functions. ReLU networks scale well to high input dimensions and can be trained on large datasets of scattered and noisy data. The MILP formulation in (4) enables the embedding of ReLU networks in optimization problems that can be solved by state-of-the-art MILP solvers.

In statistical learning it is common to follow the principle of *Occam's razor* and avoid unnecessary model complexity when searching for models that generalize well. According to this principle, deep ReLU networks are favorable to shallow networks since they can achieve the same complexity (number of linear partition regions) with fewer parameters. An example of this is given in Table 7, where shallow (2851 parameters) and deep (1801 parameters) riser networks achieve similar accuracy. Smaller networks also tend to lower solution times in the optimization, which would lead us to conclude that deep networks are favorable for surrogate modeling. Paradoxically, the deepening of networks which has yielded favorable results on modeling tasks, seems to make the resulting MILP models harder to optimize (as shown in the numerical results). Thus, what is a strength in the modeling setting, may become a weakness in the optimization setting. We are thus compelled to search for a good trade-off between model complexity and optimization efficiency. The same observation was reported in [33] for global optimization of DNNs.

Unfortunately, we observe, as others have before us, and as theory predicts, that the feasibility of using the MILP formulation quickly fades with increasing network sizes. Our numerical study indicates that the MILP formulation is unfit for applications involving large ReLU networks with thousands of hidden nodes. For the oil production optimization case, we reached a practical limit on 100 hidden nodes for the deep network architectures. Solution times are sensitive to tightness of the variable bounds in the MILP formulation, which directly translates to the big-M values in (4). Our study of bound tightening procedures, ranging from the cheap LRR procedure to the optimal NO-R procedure, shows that it is problem specific which procedure strikes the best computational efficiency. For optimization problems with small networks embedded, the cheaper procedures seem to perform best, while for larger networks the more expensive procedures perform best. Somewhat surprisingly, the MILP-based procedures, usually thought to be too computationally expensive, performs quite well on the more challenging problems. This is further amplified in applications where the optimization problem is solved many times between each time the network is updated. Here, spending the extra time in the more expensive bound tightening procedures can really pay off.

When a ReLU network is embedded in an optimization problem it is likely to be subject to output bounds. Our study is the first to investigate bound tightening of (4) in the presence of output bounds. In Table 2 we summarize the procedures and identify the ones capable of BBP. The numerical results in Section 5 show that ReLU relaxation significantly reduces bound tightness for both FBP and BBP. From the results, and the argument in Appendix A, we conclude that BBP is less effective than FBP. Of the studied procedures, only NO-R is capable of exploiting output bounds to reduce bounds; see Section 5.1. This may explain why NO-R is the best-performing procedure for the challenging oil production optimization case.

We finally remark that, while the application of (4) is practically limited to small-sized networks, many interesting nonlinearities can be approximated by networks of this size. Our case studies show that the MILP formulation of ReLU networks in (4) is an attractive approach to surrogate modeling in process optimization.

6.1. Promising research directions

Optimization of ReLU networks is currently an active area of research. Below, we highlight some promising research topics that may allow practitioners to embed larger ReLU networks in optimization problems.

- In this work we only combined LRR with the other BT procedures. Other combinations may be more efficient in terms of tightening per processing time unit. For example, by combining ITER-RR with NO-R, we may first solve a round of LP subproblems, before we invoke the more expensive NO-R for further tightening.

- The feasible region defined by the set of bounds B is a multidimensional box and all procedures presented herein only work with one ReLU at a time. It is possible to devise a procedure for simultaneous tightening of multiple bounds. For example, a procedure could tighten the sum of bounds in the same layer.

- Strong MILP formulations and a related family of cutting planes for ReLU networks were recently presented in [34]. These advancements could extend the applicability of ReLU networks as surrogate models by lowering solution times.
- Cuts and bound tightening may be used deeper in the MILP solver's B&B tree. It may be advantageous to employ cheap BT procedures specialized for the structure of ReLU network at selected branches in the branch-and-bound tree.
- L_1 regularization can be utilized to encourage sparsity in the parameter matrices, and likely reduce solution times in the optimization [34].
- Larger networks may be considered by utilizing specialized solution methods for ReLU networks. In particular, from the literature on robustness and verification of DNNs we find methods such as the modified Simplex algorithm in [35], the Lagrangian relaxation-based method in [36], and methods based on linear and Lipschitz relaxations [37, 38]. While these works focus on forward bound-propagation in a single network, their ideas may be exploited to efficiently tighten the big-M values in (4) or to develop new solution methods for problems with multiple ReLU network surrogate models.

Due to the success of ReLU networks, we believe that it is important to push the practical limitation on network size, and allow for more expressive networks as surrogate models in process optimization.

Appendix A. Forward- vs backward-propagation of bounds

Consider n nodes with outputs x_i feeding into a node y. The nodes are related as follows

$$y = \sum_{i=1}^{n} w_i x_i + b, \tag{A.1}$$

with weights $w_i \in \mathbb{R}$ and $b \in \mathbb{R}$. If x_i are outputs of ReLU units we have $l_i \geq 0$ and and no sign restrictions on w_i. Without loss of generality and to ease the presentation, we here instead require $w_i \geq 0$ and allow negative bounds on x_i, which yields an equivalent argument. A forward-propagation of bounds gives $y \in [L_f, U_f]$, with

$$L_f = \sum_{i=1}^{n} w_i l_i + b \quad \text{and} \quad U_f = \sum_{i=1}^{n} w_i u_i + b. \tag{A.2}$$

Thus, it is clear that the bounds on y are linearly dependent on the bounds of the x_i-s. A consequence of this is that a tightening of any bound u_i or l_i will have a tightening effect on U_f or L_f, respectively.

Now, consider the backward-propagation of bounds on y to node x_j. Let $y \leq U_f - \Delta U$, where $\Delta U \geq 0$ is the amount of tightening in the upper bound of y. We are interested in how this affects the bound $x_j \leq u_j - \Delta u_j$, where $\Delta u_j \geq 0$ is the change in the upper bound of u_j. Using the relationship between the nodes we have that

$$w_j x_j = y - \sum_{i \neq j} w_i x_i - b$$

The upper bound on x_j is then found to be

$$\begin{aligned} w_j(u_j - \Delta u_j) &= U_f - \Delta U - \sum_{i \neq j} w_i l_i - b \\ &= \sum_{i=1}^{n} w_i u_i + b - \Delta U - \sum_{i \neq j} w_i l_i - b \\ &= \sum_{i=1}^{n} w_i u_i - \Delta U - \sum_{i \neq j} w_i l_i \end{aligned}$$

Rearranging the last equation yields

$$\Delta u_j = \frac{\Delta U}{w_j} - \sum_{i \neq j} \frac{w_i}{w_j}(u_i - l_i)$$

Tightening of the upper bound on x_j occurs only when $\Delta u_j \geq 0$, and we obtain the requirement

$$\frac{\Delta U}{w_j} - \sum_{i \neq j} \frac{w_i}{w_j}(u_i - l_i) \geq 0$$

$$\implies \Delta U \geq \sum_{i \neq j} w_i(u_i - l_i) \geq 0 \tag{A.3}$$

An equivalent argument can be made for the lower bound $x_j \geq l_j + \Delta l_j$, given a tightening $y \geq L_f + \Delta L$. Requiring $\Delta l_j \geq 0$ and $\Delta L \geq 0$ gives

$$\Delta L \geq \sum_{i \neq j} w_i(u_i - l_i) \geq 0. \tag{A.4}$$

Inequalities (A.4) and (A.3) tell us that a considerable change in the bound of y is required for it to have any tightening effect on the bounds of preceding nodes. To better understand these requirements we may cast them as a relative change in the bounds by considering the quantity $\Delta/(U_f - L_f)$, where Δ is the change in the bound of y (either ΔU, ΔL, or a combination of these). Since $U_f - L_f \geq 0$, we may write

$$\begin{aligned}
\frac{\Delta}{U_f - L_f} &\geq \frac{1}{U_f - L_f} \sum_{i \neq j} w_i(u_i - l_i) \\
&= \frac{\sum_{i \neq j} w_i(u_i - l_i)}{\sum_{i=1}^{n} w_i(u_i - l_i)} \\
&= 1 - \frac{w_j(u_j - l_j)}{\sum_{i=1}^{n} w_i(u_i - l_i)} \\
&= 1 - \delta,
\end{aligned}$$

where we introduced $\delta \in [0, 1]$ in the last line. It now becomes evident that a relatively large change Δ is required for backward-propagation of bounds to have any effect. We see that $\delta \approx 1$ only if $w_j \gg w_i$ or $u_j - l_j \gg u_i - l_i$ for all $i \neq j$, meaning that the relationship between x_j and y dominates the relationship between y and the other nodes.

We end this discussion by considering the special case where $w_i = w_j$, $u_i = u_j$, and $l_i = l_j$ for all i, j. In this case δ simplifies to $\delta = 1/n$ and we get

$$\frac{\Delta}{U_f - L_f} \geq 1 - 1/n.$$

From this expression we see that for wide layers (large n) with "balanced" nodes, the tightening Δ must be very large for it to have any effect on preceding nodes. E.g., for $n = 100$, a reduction of 99% is required in the bounds on y.

Appendix B. Statistics for measuring bound tightness

We introduce two statistics to measure the quality of the bounds produced by a bound tightening procedure. We measure the mean absolute distance (MAD) of a set of bounds B as

$$MAD(B) := \sum_{k=0}^{K} n_k^{-1} \sum_{j=1}^{n_k} |u_j^k - l_j^k|, \tag{B.1}$$

where u_j^k and l_j^k are the upper and lower bound on $x_j^k - s_j^k$ for node (j, k).

To compare bounds, we use the mean relative distance (MRD), which measures the relative tightness of bounds B compared to the best bounds $B^\star$ and the suboptimal LRR bounds denoted B^-. The MRD is defined as follows:

$$MRD(B, B^\star, B^-) := 100 \frac{|\mathrm{MAD}(B) - \mathrm{MAD}(B^\star)|}{|\mathrm{MAD}(B^-) - \mathrm{MAD}(B^\star)|}. \tag{B.2}$$

For a set of optimal bounds $B = B^*$, $MRD(B, B^\star, B^-) = 0$. While for bounds $B = B^-$, the MRD is 100. The bound tightening procedures used in this study are initialized with B^0, and thus always obtain an MRD in the range $[0, 100]$. Note that we define the MRD to be zero whenever the denominator is zero.

Appendix C. Iterative RR procedure

Algorithm 2 ITER-RR

```
Require: Parameters $\Theta$, initial bounds $B^-$, MAD ratio threshold $\epsilon \in (0,1)$,
  max iterations $N \geq 1$
  $B \leftarrow B^-$
  $m^- = \mathrm{MAD}(B^-)$
  $r = 0$
  $i \leftarrow 0$
  while $r \leq \epsilon$ and $i \leq N$ do
      $B \leftarrow \mathrm{RR}(\Theta, B)$                ▷ Run RR procedure
      $m \leftarrow \mathrm{MAD}(B)$
      $r \leftarrow m/m^-$                              ▷ Compute ratio
      $m^- \leftarrow m$
      $i \leftarrow i + 1$
  end while
  return Tightened bounds $B$
```

Appendix D. Bound tightening results

Table D.10: Bound tightening results for Q_n. The reported values are the average results over ten models, for $n = 1, \dots, 6$.

n	BT procedure	Time (s)	Dead neurons (%)	MAD	MRD (%)
1	LRR	0.013	29.3	2.075	100.0
	RR	0.024	41.3	1.270	9.4
	ITER-RR	0.049	42.7	1.192	2.7
	LR(1)	0.054	32.7	1.695	39.2
	SEMI-LR(1)	0.138	42.0	1.238	4.5
	NO-R(1)	0.070	43.3	1.179	0.0
2	LRR	0.024	9.3	5.832	100.0
	RR	0.072	12.0	2.972	23.0
	ITER-RR	0.232	14.3	2.730	15.7
	LR(1)	0.374	12.3	2.749	15.2
	SEMI-LR(1)	1.575	12.7	2.598	10.7
	NO-R(1)	1.959	16.7	2.228	0.0
3	LRR	0.065	9.0	9.879	100.0
	RR	0.238	9.2	5.046	30.2
	ITER-RR	0.793	10.2	4.665	23.4
	LR(1)	3.772	10.2	3.866	12.4
	SEMI-LR(1)	17.42	10.5	3.747	9.6
	NO-R(1)	36.65	13.8	3.245	0.0
4	LRR	0.082	4.9	14.76	100.0
	RR	0.315	4.9	6.72	26.6
	ITER-RR	0.841	4.9	6.40	23.5
	LR(1)	7.717	6.1	4.75	7.6
	SEMI-LR(1)	32.02	6.1	4.68	6.8
	NO-R(1)	84.29	8.9	4.13	0.0
5	LRR	0.164	4.0	67.47	100.0
	RR	1.174	4.1	24.04	27.3
	ITER-RR	5.245	4.1	17.58	16.3
	LR(1)	95.59	7.5	8.79	1.6
	SEMI-LR(1)	181.2	6.9	9.18	2.2
	NO-R(1)	303.9	7.0	8.55	1.1
6	LRR	0.286	3.4	48.18	100.0
	RR	3.174	3.4	17.87	26.3
	ITER-RR	14.44	3.4	12.91	14.0
	LR(1)	195.8	4.2	7.14	0.0
	SEMI-LR(1)	335.0	3.4	9.59	6.1
	NO-R(1)	472.9	3.5	8.49	3.4

Table D.11: Bound tightening results for shallow well networks with layers (1, 20, 20, 1). The reported values are the average results of the eight models.

BT method	Time (s)	Dead neurons (%)	MAD	MRD (%)
LRR	0.020	0.0	0.21916	100.0
RR	0.042	1.8	0.20278	12.3
ITER-RR	0.068	1.8	0.20170	3.2
LR(60)	0.097	1.8	0.20314	14.6
SEMI-RR(60)	0.548	1.8	0.20273	12.0
NO-R(60)	0.104	1.8	0.20138	0.0
NO-R	0.104	1.8	0.20138	0.0

Table D.12: Bound tightening results for deep well networks with layers (1, 10, 10, 10, 10, 1). The reported values are the average results of the eight models.

BT method	Time (s)	Dead neurons (%)	MAD	MRD (%)
LRR	0.022	0	0.57230	100.0
RR	0.039	0	0.54579	11.7
ITER-RR	0.062	0	0.54313	3.2
LR(60)	0.183	0	0.54610	12.5
SEMI-RR(60)	0.503	0	0.54568	11.5
NO-R(60)	0.156	0	0.54213	0.0
NO-R	0.156	0	0.54213	0.0

Table D.13: Bound tightening results for shallow riser networks with layers (4, 50, 50, 1). The reported values are the average results of the two models.

BT method	Time (s)	Dead neurons (%)	MAD	MRD (%)
LRR	0.093	4.0	2.91520	100.0
RR	0.460	7.0	2.00257	46.3
ITER-RR	0.991	7.0	1.98497	45.1
LR(60)	8.889	16.0	1.57080	27.2
SEMI-RR(60)	55.29	16.0	1.52058	23.6
NO-R(60)	306.0	19.5	1.12196	0.0
NO-R	306.0	19.5	1.12196	0.0

Table D.14: Bound tightening results for deep riser networks with layers (4, 20, 20, 20, 20, 20, 1). The reported values are the average results of the two models.

BT method	Time (s)	Dead neurons (%)	MAD	MRD (%)
LRR	0.061	0	13.4183	100.0
RR	0.546	0	6.31994	44.4
ITER-RR	5.425	1	3.83465	25.2
LR(60)	109.3	4	2.44874	14.7
SEMI-RR(60)	134.9	4	2.37722	14.2
NO-R(60)	1771	11	0.67823	0.1
NO-R	5080	11	0.66211	0.0

[1] D. R. Jones, M. Schonlau, W. J. Welch, Efficient global optimization of expensive black-box functions, Journal of Global Optimization 13 (4) (1998) 455 – 492.

[2] M. Ghavamzadeh, S. Mannor, J. Pineau, A. Tamar, et al., Bayesian reinforcement learning: A survey, Foundations and Trends in Machine Learning 8 (5-6) (2015) 359 – 483.

[3] I. Fahmi, S. Cremaschi, Process synthesis of biodiesel production plant using artificial neural networks as the surrogate models, Computers & Chemical Engineering 46 (2012) 105 – 123.

[4] R. G. Regis, C. A. Shoemaker, A Stochastic Radial Basis Function Method for the Global Optimization of Expensive Functions, INFORMS Journal on Computing 19 (4) (2007) 497 – 509.

[5] H. R. Sant Anna, A. G. Barreto, F. W. Tavares, M. B. de Souza, Machine learning model and optimization of a psa unit for methane-nitrogen separation, Computers & Chemical Engineering 104 (2017) 377 – 391.

[6] F. Ye, S. Ma, L. Tong, J. Xiao, P. Bénard, R. Chahine, Artificial neural network based optimization for hydrogen purification performance of pressure swing adsorption, International Journal of Hydrogen Energy 44 (11) (2019) 5334 – 5344.

[7] B. Grimstad, B. Foss, R. Heddle, M. Woodman, Global optimization of multiphase flow networks using spline surrogate models, Computers & Chemical Engineering 84 (2016) 237 – 254.

[8] J. P. Vielma, S. Ahmed, G. Nemhauser, Mixed-Integer Models for Nonseparable Piecewise-Linear Optimization: Unifying Framework and Extensions, Operations Research 58 (2) (2010) 303 – 315.

[9] N. Martinez, H. Anahideh, J. M. Rosenberger, D. Martinez, V. C. Chen, B. P. Wang, Global optimization of non-convex piecewise linear regression splines, Journal of Global Optimization 68 (3) (2017) 563 – 586.

[10] A. N. Elmachtoub, P. Grigas, Smart "Predict, then Optimize" (2017). `arXiv:1710.08005`.

[11] V. V. Mišić, Optimization of Tree Ensembles (2017). `arXiv:1705.10883`.

[12] M. Biggs, R. Hariss, Optimizing Objective Functions Determined from Random Forests, SSRN Electronic Journal (2017) 1 – 49.

[13] R. Balestriero, R. Baraniuk, Mad Max: Affine Spline Insights into Deep Learning (2018). `arXiv:1805.06576`.

[14] R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, M. P. Kumar, A Unified View of Piecewise Linear Neural Network Verification (2017). `arXiv:1711.00455`.

[15] C.-H. Cheng, G. Nührenberg, H. Ruess, Maximum resilience of artificial neural networks, in: D. D'Souza, K. Narayan Kumar (Eds.), Automated Technology for Verification and Analysis, Springer International Publishing, Cham, 2017, pp. 251 – 268.

[16] S. Dutta, S. Jha, S. Sanakaranarayanan, A. Tiwari, Output Range Analysis for Deep Feedforward Neural Networks (2017). `arXiv:1709.09130`.

[17] V. Tjeng, K. Xiao, R. Tedrake, Evaluating Robustness of Neural Networks with Mixed Integer Programming (2017). `arXiv:1711.07356`.

[18] M. Fischetti, J. Jo, Deep neural networks and mixed integer linear optimization, Constraints 23 (3) (2018) 296 – 309.

[19] T. Serra, C. Tjandraatmadja, S. Ramalingam, Bounding and Counting Linear Regions of Deep Neural Networks (2018). `arXiv:1711.02114`.

[20] A. Kumar, T. Serra, S. Ramalingam, Equivalent and Approximate Transformations of Deep Neural Networks (2019). `arXiv:1905.11428`.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, Software available from tensorflow.org (2015).
URL `https://www.tensorflow.org/`

[22] R. B. Hughes, M. R. Anderson, Simplexity of the cube, Discrete Mathematics 158 (1-3) (1996) 99 – 150.

[23] R. Misener, C. A. Floudas, Piecewise-linear approximations of multidimensional functions, Journal of Optimization Theory and Applications 145 (1) (2010) 120 – 147.

[24] K. Crombecq, D. Gorissen, D. Deschrijver, T. Dhaene, A novel hybrid sequential design strategy for global surrogate modeling of computer experiments, SIAM Journal of Scientific Computing 33 (2011) 1948 – 1974.

[25] J. van der Herten, I. Couckuyt, D. Deschrijver, T. Dhaene, A fuzzy hybrid sequential design strategy for global surrogate modeling of high-dimensional computer experiments, SIAM Journal of Scientific Computing 37 (2015) 1020 – 1039.

[26] D. Gorissen, I. Couckuyt, E. Laermans, Multiobjective global surrogate modeling, dealing with the 5-percent problem, Engineering with Computers 26 (2010) 81 – 98.

[27] E. M. L. Beale, J. A. Tomlin, Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables, in: J. Lawrence (Ed.), Proceedings of the fifth international conference on operational research, 1970, pp. 447 – 454.

[28] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, J. Sohl-Dickstein, On the Expressive Power of Deep Neural Networks (2016). `arXiv:1606.05336`.

[29] L. Bottou, F. E. Curtis, J. Nocedal, Optimization Methods for Large-Scale Machine Learning, SIAM Review 60 (2) (2018) 223 – 311.

[30] A. M. Gleixner, T. Berthold, B. Müller, S. Weltge, Three enhancements for optimization-based bound tightening, Journal of Global Optimization 67 (4) (2017) 731 – 757.

[31] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual (2018).
URL `http://www.gurobi.com`

[32] K. He, X. Zhang, S. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, in: 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026 – 1034.

[33] A. M. Schweidtmann, A. Mitsos, Global Deterministic Optimization with Artificial Neural Networks Embedded (2018). `arXiv:1801.07114`.

[34] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, J. P. Vielma, Strong mixed-integer programming formulations for trained neural networks (2018). `arXiv:1811.01988`.

[35] G. Katz, C. Barrett, D. Dill, K. Julian, M. Kochenderfer, Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks (2017). `arXiv:1702.01135`.

[36] Krishnamurthy, Dvijotham, R. Stanforth, S. Gowal, T. Mann, P. Kohli, A Dual Approach to Scalable Verification of Deep Networks (mar 2018). `arXiv:1803.06567`.

[37] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, L. Daniel, Towards Fast Computation of Certified Robustness for ReLU Networks (2018). `arXiv:1804.09699`.

[38] G. Singh, T. Gehr, M. Mirman, M. Püschel, M. Vechev, Fast and Effective Robustness Certification, Advances in Neural Information Processing Systems 31 (NIPS 2018) (2018) 10802 – 10813.