Supervised learning of convex piecewise linear approximations of optimization problems

Laurine Duchesne, Quentin Louveaux and Louis Wehenkel

University of Liege - Dept of EE&CS
Liege - Belgium

Abstract. We propose to use input convex neural networks (ICNN) to build convex approximations of non-convex feasible sets of optimization problems, in the form of a set of linear equalities and inequalities in a lifted space. Our approach may be tailored to yield both inner- and outer- approximations, or to maximize its accuracy in regions closer to the minimum of a given objective function. We illustrate the method on two-dimensional toy problems and motivate it by various instances of reliability management problems of large-scale electric power systems.

1 Introduction

Mathematical optimization is a very rich framework allowing us to model lots of practical problems. The tractability of an optimization problem depends on the properties of the objective function and the feasible set. Non-convex problems are often intractable whereas convex problems are tractable. In particular, linear optimization, which is a subclass of convex problems, is a mature field where problems with thousands of variables and constraints can be routinely solved with efficient and reliable solvers [1].

In this paper, we propose to develop machine learning approaches that would allow us to automatically build convex approximations of non-linear and/or non-convex feasible domains in the form of a set of linear (and thus convex) constraints, in order to exploit the extremely efficient methods and solvers already available for linear programs. If the obtained linear approximation of the feasible set is an inner approximation (i.e. all the solutions belonging to this approximation are feasible), it would allow us to generate feasible solutions and, in the case of a convex objective function, upper bounds of the minimal value of the original problem. If it is an outer approximation (i.e. the approximation contains all the feasible solutions of the original problem) it would provide lower bounds, also in the case of a convex objective function. Note that this constraint on the objective function is not restrictive because generically, any optimization problem $\min_x f(x)$ s.t. $x \in \mathcal{D}$ may be rewritten as $\min_{(x,z)} z$ s.t. $z \geq f(x)$, $x \in \mathcal{D}$ so that any non-linear and/or non-convex optimization problem may be reduced to the minimization of a linear (thus convex) function subject to a possibly non-convex feasible set.

The ICNN [2] is a neural network with constraints on its parameters and activation functions implying that the learnt input-output function $h(x, \theta)$ is a convex function of the inputs x. While this method was originally proposed in a regression context (e.g. to build convex approximations of objective functions of

optimization problems), we propose to use it in a classification setting in order to build convex approximations of feasible sets of optimization problems. Note that this approach can also be of interest in the context of convex feasible sets. Using an ICNN would indeed allow us to enforce the convexity property when using supervised learning to approximate a convex feasible set.

The rest of this paper is organized as follows. Section 2 presents the main idea, i.e. the feed-forward ICNN model and our proposed adaptation to represent and learn convex approximations of a feasible domain, and how this learnt convex approximation can be used effectively in an optimization problem if the ICNN architecture is using piecewise linear activation functions such as ReLU or leaky-ReLU. Section 3 presents some illustrative experiments, section 4 discusses related works, and section 5 possible real-world applications and directions for future work. Moreover, we provide an appendix which gives further details about simulation experiments and mathematical proofs.

2 ICNNs for convex classification and optimization

We consider feed-forward networks as shown in Fig. 1, where $x \in \mathbb{R}^n$ denotes the vector of inputs, $\theta = \{W_i^z, W_i^x, b_i\}_{i=0,...,k-1}$ the set of parameters, and g_i the activation function used in the ith layer. The relationship between the inputs and the outputs of layer i of such a model is thus recursively given by:

$$z_{i+1} = g_i(W_i^z \times z_i + W_i^x \times x + b_i) \text{ for } i = 0, ..., k-1,$$

with $z_0 = 0$ and (the outputs) $h(x, \theta) = z_k$.

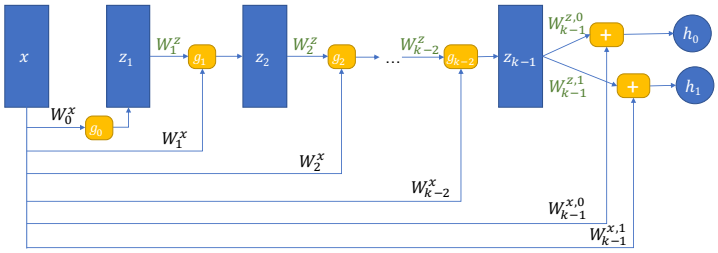


Fig. 1: Representation of the layers of an ICNN with two outputs h_0 and h_1. The weights W^z, colored in green, are constrained to be non-negative.

Notice that compared to a classical feed-forward neural network, *pass-through layers* connecting directly the input vector x to each layer have been added to increase the representation power of these networks. In the ICNN model [2], the weights W_i^z for $i = 1, ..., k-1$ are constrained to be non-negative and the activation functions g_i are constrained to be convex and non-decreasing. These two conditions are sufficient to guarantee that the activations z_i of each layer and hence the outputs $h(x, \theta)$ are convex functions of the input vector x. In the rest of this paper we will use (convex and non-decreasing) *piecewise linear* activation functions g_i (such as ReLU or leaky-ReLU).

2.1 Convex set representation by an ICNN with two outputs

Among several possibilities, we decided to use an ICNN with two (scalar) outputs h_0 and h_1 to create a binary classifier, where an input is associated to the target class 0 if $g(x,\theta) = h_1(x,\theta) - h_0(x,\theta) \leq 0$.

With this choice, it is clear that the set $\tilde{\mathcal{D}}$ of elements classified in class 0 by the ICNN is a convex subset of $\mathbb{R}^n$, as soon as $g(x,\theta)$ is a convex function of x. In order to ensure this, we use identity activation functions for the output layer ($g_{k-1}(x) = x$) and impose an additional constraint on the parameter vectors $W_{k-1}^{z,0}$ and $W_{k-1}^{z,1}$ feeding the output layer of the ICNN: they should satisfy component-wise the inequality

$$W_{k-1}^{z,1} \geq W_{k-1}^{z,0}. \tag{1}$$

Notice that if we feed such a network with an extended vector $x_e = (x, -x)$ (or, more generally $x_e = Ax$), the input-output relationship remains convex in x.

2.2 Building a family of nested convex sets

To build a convex approximation of a set $\mathcal{D} \subset \mathbb{R}^n$, we assume that we have (or that we can build) a dataset of input-output pairs $\hat{\mathcal{D}} = \{x^i, y^i\}_{i=1}^n$, where each input x^i describes the coordinates of a point in $\mathbb{R}^n$ and where the corresponding output $y^i = 0$ if the point x^i belongs to $\mathcal{D}$ and $y^i = 1$ otherwise.

We propose to learn from the dataset $\hat{\mathcal{D}}$ the parameters θ of an ICNN classifier which has as inputs $x_e = (x, -x)$, and by using the cross-entropy loss

$$\text{loss}(\theta, x, y) = -\log\left(\frac{\exp(h_y(\theta,x))}{\exp(h_0(\theta,x)) + \exp(h_1(\theta,x))}\right).$$

After training, we consider the whole family of (convex) sets

$$\tilde{\mathcal{D}}_\lambda = \{x \in \mathbb{R}^n | g(x,\theta) = h_1(x,\theta) - h_0(x,\theta) \leq \lambda\},$$

with $\lambda \in \mathbb{R}$, as candidate convex approximations of $\mathcal{D}$.

2.3 Exploitation in the context of optimization

A convex ICNN classifier can be used to approximate the feasible set $\mathcal{D}$ of an optimization problem. If the objective $f(x)$ is convex, the approximated problem

$$\min_{x \in \tilde{\mathcal{D}}_\lambda} f(x), \tag{2}$$

is then also convex. Furthermore, if the objective function $f(x)$ is (piecewise) linear (and convex), and if all the activation functions g_i used in the ICNN are piecewise linear, convex, and non-decreasing functions (such as $ReLU(x) = \max(0, x)$, or $leaky - ReLU(x) = \max(0.01x, x)$), we can show that the resulting optimization problem reduces to a linear program (see section B of the appendix). In general, we can use convex ICNN classifiers to approximate in a linear fashion any non-convex part of the constraints and/or objective function of any optimization problem. Solving (2) for an increasing sequence of λ values, would yield a decreasing sequence of optimal values.

3 Illustrations

We consider some (convex and non-convex) toy problems where $\mathcal{D} \subset \mathbb{R}^2$. For each one, we used a dataset of 20,000 labelled points. These points were sampled uniformly in a square of length 10 centered at $(0, 0)$; 16,000 were used to train and 4,000 to test. We show results with ICNNs of 6 hidden layers and 50 neurons per layer, and ReLU activations. The ADAM optimizer [3] with a learning rate of 10^{-3} was used to update the network parameters at each epoch. To enforce the non negativity condition for the weights W^z, each negative element of the update computed with the optimizer is set to 0 before the next iteration. Similarly, to enforce the convexity constraint on the last layer, we set to 0 each pair of weights for which the convexity condition (1) is not met. Before training, the inputs are standardized based on their minimum and maximum values in the training set to be in the range $[0, 1]$.

Fig. 2 shows four regions $\mathcal{D}$ and their approximation $\tilde{\mathcal{D}}_0$ with an ICNN. One can see that the convex region is well approximated by the ICNN. For the non-convex domains, the ICNN provides a convex approximation $\tilde{\mathcal{D}}_0$ of $\mathcal{D}$.

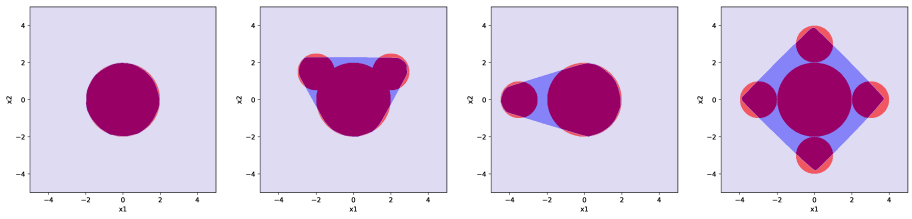
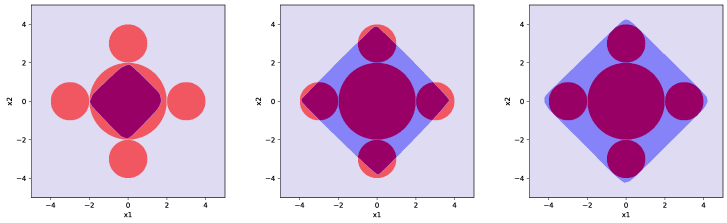


Fig. 2: The set $\mathcal{D}$ is represented in red and the approximated set $\tilde{\mathcal{D}}_0$ in blue.



(a) $\lambda = -1.75$ (b) $\lambda = 0$ (c) $\lambda = 2.7$

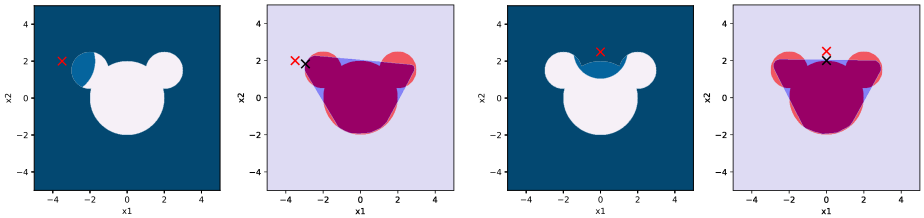
Fig. 3: Effect of λ on the $\tilde{\mathcal{D}}_\lambda$ set.

It is possible to play with the size of the approximated region by modifying the threshold λ in the definition $\tilde{\mathcal{D}}_\lambda = \{x \in \mathbb{R}^n | h_1(x) - h_0(x) \leq \lambda\}$. Fig. 3 shows the approximated region $\tilde{\mathcal{D}}_\lambda$ for various λ. We see that increasing the threshold allows us to find outer approximations of $\mathcal{D}$ and decreasing the threshold allows us to find inner approximations. With this method, the model needs only to be learnt once and then the threshold can be manually adjusted to obtain inner or outer approximations.

3.1 Considering an objective function when learning the ICNN

In an optimization context, it is of interest to guide the learning of the ICNN in order to improve the approximation close to the minimum values of the objective function. For that, one possibility is to exploit the objective function $f(x)$ in the definition of the loss function used when training the classifier, by giving less weight to elements of the training set farther from the optimum, and thus induce the learnt approximation $\tilde{\mathcal{D}}_0$ to be tighter near to the sought optimum. More details about this procedure can be found in section C of the appendix.

Fig. 4 shows decision boundaries thus obtained for different objective functions. We see that the approximation is indeed tighter close to the unconstrained optimum and actually approaches the constrained optimum very well.



(a) $f(x) = (x_1 + 3.5)^2 + (x_2 - 2)^2$ (b) $f(x) = (x_1)^2 + (x_2 - 2.5)^2$

Fig. 4: Left parts of (a) and (b): training weights $w_{y,f}$ (blue/dark blue: $w_{y,f} = 1$; white: $w_{y,f} = 0.2$). Right parts: resulting $\tilde{\mathcal{D}}_0$, where red and black crosses indicate respectively the unconstrained and the constrained minimum of $f(x)$.

4 Related works

This work is not the first one to exploit the particular properties of ICNNs in the context of optimization problems. ICNNs have been used for complex physical systems control [4, 5, 6], to learn the objective function of an optimization problem and/or its constraints. In these papers, the considered case-studies are convex. Using ICNNs is therefore a way to exploit the prior knowledge of convexity while offering tractable control methods.

Similar to our method, ICNNs are used [7] in the context of non-convex optimization. The authors developed an algorithm called the Convex Difference Neural Network that expresses the learnt function as a difference of convex functions, so that they can use Difference of Convex programming techniques in problems where their algorithm has been used to learn objective functions and constraints of optimization problems.

Compared to this research, our method learns convex (actually linear) approximations of non-convex optimization problems, at the cost of possibly larger approximation errors but with the advantage that much more efficient and scalable optimization solvers can be used.

5 Conclusions and future works

We propose to use ICNNs to learn convex approximations of feasible sets of general optimization problems. This approximation reduces to a series of linear inequalities when ReLU activation functions are used, and we showed how the model may yield outer or inner approximations and/or tight approximations in regions near the optimum of a given objective function.

The next step is to test this method on practical non-convex optimization problems. Depending on the context in which this method could be applied, the found solution could be used directly or as a warm-start point for solving the non-convex problem. It can also be used to compute lower or upper bounds of the optimal solution when the learnt approximation is built so as to obtain an outer or an inner approximation of the feasible region.

Another direction of research is to consider the case where the feasible set $\mathcal{D}$ to be approximated depends on some external parameters ξ. In electric power systems, for instance, the secure region of operation depends on load and renewable generation levels and so ξ could represent the realizations of these exogenous uncertainties. One could then learn a convex approximation of the parameterized domain $\mathcal{D}(\xi)$ with an ICNN that would have both x and ξ as inputs while being constrained to be convex only in x (in [2] such models are called *Partially* Input Convex NNs). If the ICNN is able to capture the relationship between the shape of the feasible region and the parameters ξ, then it could largely speed up the solving of this type of problems with varying ξ values, once it is learnt.

A further direction of research would consider distributed optimization problems, where the proposed approach could be used to enable various agents to learn and exchange convex approximations of their subsets of constraints and their sub-objectives. This would be of extremely high relevance to the field of multi-area electric power systems planning and operation.

References

[1] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[2] Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Yize Chen, Yuanyuan Shi, and Baosen Zhang. Optimal control via neural networks: A convex approach. *arXiv preprint arXiv:1805.11835*, 2018.

[5] Yize Chen, Yuanyuan Shi, and Baosen Zhang. Data-driven optimal voltage regulation using input convex neural networks. *Electric Power Systems Research*, 189:106741, 2020.

[6] Shu Yang and B Wayne Bequette. Optimization-based control using input convex neural networks. *Computers & Chemical Engineering*, 144:107143, 2021.

[7] Parameswaran Sankaranarayanan and Raghunathan Rengaswamy. CDiNN-convex difference neural networks. *arXiv preprint arXiv:2103.17231*, 2021.

A Introduction

This appendix is organized as follows.

In section B, we provide the mathematical proof that the approximated problem

$$\min\{f(x)\} \text{ s.t. } x \in \tilde{\mathcal{D}}_\lambda,$$

with $f(x)$ piecewise linear and convex and $\tilde{\mathcal{D}}_\lambda$ a convex feasible set approximation built according to the method described, can be written as a linear program, in the case of piecewise linear, convex, and non-decreasing activation functions.

In section C, we describe how we adapted the loss function of the ICNN in order to improve the quality of the approximation in regions close to the optimum.

B Proof: $\min_{x \in \tilde{\mathcal{D}}_\lambda} f(x)$ can be reduced to a linear program

We consider a convex ICNN classifier used to approximate the feasible set $\mathcal{D}$ of an optimization problem. Let us show that if the objective function $f(x)$ is piecewise linear and convex, and if all the activation functions g_i used in the ICNN are piecewise linear, convex, and non-decreasing functions (such as $ReLU(x) = \max(0, x)$, or $leaky - ReLU(x) = \max(0.01x, x)$), the resulting optimization problem

$$\min\{f(x)\} \text{ s.t. } x \in \tilde{\mathcal{D}}_\lambda,$$

reduces to a linear program. We first notice that if $f(x)$ is piecewise linear and convex, then

$$\min\{f(x)\} \text{ s.t. } x \in \tilde{\mathcal{D}}_\lambda$$

may also be rewritten as

$$\min\{z\} \text{ s.t. } z \geq a_j^T x + b_j, \forall j = 1, \ldots, l; x \in \tilde{\mathcal{D}}_\lambda,$$

where the set of inequalities $z \geq a_j^T x + b_j, \forall j = 1, \ldots, l$ represent the epigraph of $f(x)$. We thus need only to prove that $\tilde{\mathcal{D}}_\lambda$ may itself also be represented by a set of linear (in)equalities.

For simplicity, we prove this in the case of ReLU activation functions but the result can be extended to any other choice of piecewise linear, convex and non-decreasing activation functions.

First, let us consider the domain P which is the set of points $(x, z_0, \ldots, z_{k-1})$ such that

$$z_0 = 0, \tag{3}$$

and

$$\forall i = 0, \ldots k-2 : z_{i+1} = \max(W_i^z \times z_i + W_i^x \times x + b_i, 0) \tag{4}$$

and

$$(W_{k-1}^{z,1} - W_{k-1}^{z,0}) \times z_{k-1} + (W_{k-1}^{z,1} - W_{k-1}^{x,0}) \times x + b_{k-1}^1 - b_{k-1}^0 \leq \lambda. \tag{5}$$

Note that the projection on x of the domain P, $proj_x(P)$, corresponds to $\tilde{\mathcal{D}}_\lambda$. When the max function is replaced with a set of equations, the equivalent formulation of the constraints in P becomes

$$(W_{k-1}^{z,1} - W_{k-1}^{z,0}) \times z_{k-1} + (W_{k-1}^{z,1} - W_{k-1}^{x,0}) \times x + b_{k-1}^1 - b_{k-1}^0 \leq \lambda, \tag{6}$$
$$z_{i+1} = 0 + s_i^0 \qquad \text{for } i = 0, \dots, k-2 \tag{7}$$
$$z_{i+1} = W_i^z \times z_i + W_i^x \times x + b_i + s_i^z \qquad \text{for } i = 0, \dots, k-2 \tag{8}$$
$$s_i^0 s_i^z = 0 \qquad \text{for } i = 0, \dots, k-2 \tag{9}$$
$$s_i^0, s_i^z \geq 0 \qquad \text{for } i = 0, \dots, k-2 \tag{10}$$
$$z_0 = 0, \tag{11}$$

where we introduce slack variables $s_i^\cdot$ to express that z_{i+1} is either equal to 0 or to $W_i^z \times z_i + W_i^x \times x + b_i$ for $i = 0, \dots, k-2$.

All the constraints in P are linear, except equation (9). However, we can show that this nonlinear equation is not necessary regarding our purpose because the projection on x of a relaxed version of the domain P, that we call Q and for which all the constraints defining Q correspond to a set of linear equations, is also equal to $\tilde{\mathcal{D}}_\lambda$.

Lemma 1. *We are given the parameters of an ICNN using ReLU as activation functions and learnt to build a convex approximation $\tilde{\mathcal{D}}_\lambda$ of the feasible set $\mathcal{D}$. Consider the set P defined as*

$$P = \big\{(x, z_1, \dots, z_{k-1}, s_0^0, \dots, s_{k-2}^0, s_0^z, \dots, s_{k-2}^z) \big|$$
$$(W_{k-1}^{z,1} - W_{k-1}^{z,0}) \times z_{k-1} + (W_{k-1}^{z,1} - W_{k-1}^{x,0}) \times x + b_{k-1}^1 - b_{k-1}^0 \leq \lambda, \tag{12}$$
$$z_{i+1} = 0 + s_i^0 \qquad \forall i = 0, \dots, k-2, \tag{13}$$
$$z_{i+1} = W_i^z \times z_i + W_i^x \times x + b_i + s_i^z \qquad \forall i = 0, \dots, k-2, \tag{14}$$
$$s_i^0 s_i^z = 0 \qquad \forall i = 0, \dots, k-2, \tag{15}$$
$$s_i^0, s_i^z \geq 0 \qquad \forall i = 0, \dots, k-2, \tag{16}$$
$$z_0 = 0\big\}. \tag{17}$$

The set Q, which is defined as P but where constraint (15) is removed, i.e. defined as

$$Q = \{(x, z_1, \dots, z_{k-1}, s_0^0, \dots, s_{k-2}^0, s_0^z, \dots, s_{k-2}^z) \big| \tag{18}$$
$$(W_{k-1}^{z,1} - W_{k-1}^{z,0}) \times z_{k-1} + (W_{k-1}^{z,1} - W_{k-1}^{x,0}) \times x + b_{k-1}^1 - b_{k-1}^0 \leq \lambda, \tag{19}$$
$$z_{i+1} = 0 + s_i^0 \qquad \forall i = 0, \dots, k-2, \tag{20}$$
$$z_{i+1} = W_i^z \times z_i + W_i^x \times x + b_i + s_i^z \qquad \forall i = 0, \dots, k-2, \tag{21}$$
$$s_i^0, s_i^z \geq 0 \qquad \forall i = 0, \dots, k-2, \tag{22}$$
$$z_0 = 0\}, \tag{23}$$

is therefore a relaxation of P. We now prove that the projection on x of the set P is equal to the projection on x of the set Q:

$$proj_x(P) = proj_x(Q). \tag{24}$$

Proof. 1. $proj_x(P) \subseteq proj_x(Q)$ is obvious since Q is a relaxation of P.

2. We now prove $proj_x(P) \supseteq proj_x(Q)$. Consider $(x, z, s^0, s^z) \in Q$. If $(x, z, s^0, s^z) \in P$, the result follows. Assume now that $(x, z, s^0, s^z) \notin P$. It is always possible, by following Algorithm 1, to build from a set of points (x, z, s^0, s^z) in Q but not in P, a set of points $(x, \bar{z}, \bar{s}^0, \bar{s}^z)$ in P with the same x, which proves the result.

Algorithm 1: Update $(x, z, s^0, s^z) \in Q$ such that $(x, \bar{z}, \bar{s}^0, \bar{s}^z) \in P$.

Result: $(x, \bar{z}, \bar{s}^0, \bar{s}^z)$ in P
// Initialization
$(x, \bar{z}, \bar{s}^0, \bar{s}^z) = (x, z_1, \ldots, z_{k-1}, s_0^0, \ldots, s_{k-2}^0, s_0^z, \ldots, s_{k-2}^z)$;
for $j = 0 : k-2$ **do**
  if $\bar{s}_j^0 > 0$ ***and*** $\bar{s}_j^z > 0$ **then**
    $\Delta := \min(\bar{s}_j^0, \bar{s}_j^z)$;
    $\hat{s}_j^0 = \bar{s}_j^0 - \Delta$;
    $\hat{s}_j^z = \bar{s}_j^z - \Delta$;
    $\hat{z}_{j+1} = 0 + \hat{s}_j^0$;
    // or $\hat{z}_{j+1} = W_j^z \times \bar{z}_j + W_j^x \times x + b_j + \hat{s}_j^z$
    if $j < k-2$ **then**
      $\hat{s}_{j+1}^z = \bar{s}_{j+1}^z + W_{j+1}^z(\bar{z}_{j+1} - \hat{z}_{j+1})$;
      $(\bar{z}_{j+1}, \bar{s}_j^0, \bar{s}_j^z, \bar{s}_{j+1}^z) = (\hat{z}_{j+1}, \hat{s}_j^0, \hat{s}_j^z, \hat{s}_{j+1}^z)$;
    else
      $(\bar{z}_{j+1}, \bar{s}_j^0, \bar{s}_j^z) = (\hat{z}_{j+1}, \hat{s}_j^0, \hat{s}_j^z)$;
    end
  end
end

Let us show that $(x, \bar{z}, \bar{s}^0, \bar{s}^z)$, built from $(x, z, s^0, s^z) \in Q$ with Algorithm 1, belongs to P. For that we proceed iteratively and we show that at each iteration, the updated vector still belongs to Q. At the end of the iterations, since by construction $(x, \bar{z}, \bar{s}^0, \bar{s}^z)$ satisfies constraint (15), $(x, \bar{z}, \bar{s}^0, \bar{s}^z) \in P$.

Let j be the smallest index such that $s_j^0 > 0$ and $s_j^z > 0$.

Consider the point $(x, \bar{z}, \bar{s}^0, \bar{s}^z)$, obtained after j iterations with Algorithm 1. We can readily see that this point belongs to Q. Indeed, compared to the point $(x, z, s^0, s^z) \in Q$, the only constraint with a different realization of the left-hand-side or right-hand-side is constraint (21) for $i = j$ and $i = j + 1$. The constraint is nevertheless still valid in both cases:

- $i = j$: By definition of $\bar{z}_{j+1}$, the constraint holds with equality.
- $i = j + 1$: We have that $\bar{z}_{j+1} < z_{j+1}$ since $\bar{s}_j^z < s_j^z$ by construction. Furthermore, given that $W_{j+1}^z > 0$, $W_{j+1}^z \times z_{j+1} > W_{j+1}^z \times \bar{z}_{j+1}$.

Therefore,

$$z_{j+2} \geq W^z_{j+1} \times z_{j+1} + W^x_{j+1} \times x + b_{j+1} > W^z_{j+1} \times \bar{z}_{j+1} + W^x_{j+1} \times x + b_{j+1}$$

and the constraint holds.

Note that in case $j = k-2$, the right-hand-side of constraint (19) is also impacted. However, since $(W^{z,1}_{k-1} - W^{z,0}_{k-1}) \geq 0$,

$$(W^{z,1}_{k-1} - W^{z,0}_{k-1})\bar{z}_{k-1} < (W^{z,1}_{k-1} - W^{z,0}_{k-1}) z_{k-1} \leq \lambda,$$

where the last inequality holds because $(x, z, s^0, s^z) \in Q$. Therefore, $(x, \bar{z}, \bar{s}^0, \bar{s}^z) \in Q$.

Observe that, if $(x, \bar{z}, \bar{s}^0, \bar{s}^z) \notin P$, there exists $\bar{j} > j$ such that $s^0_{\bar{j}} > 0$ and $s^z_{\bar{j}} > 0$. We can again decrease the value of $z_{\bar{j}+1}$ in order to make one of the slacks tight. We obtain the result by applying the procedure repeatedly. Since there is a finite number of indices, the procedure can be applied at most $k-1$ times until we obtain $(x, \bar{z}, \bar{s}^0, \bar{s}^z) \in P$.

We can thus state that $proj_x(P) = proj_x(Q)$. □

Note that this lemma can be extended to other activation functions, as long as they are convex, piecewise linear and non-decreasing. It can thus be applied to leaky-ReLU activation functions. It is also still valid at the limit, for an infinite number of pieces and so it is valid for any smooth convex and non-decreasing activation function.

Consequently to Lemma 1, given an objective function that only depends on x, we have the following result.

Corollary 2. *Given an ICNN using ReLU as activation functions and learnt to build a convex approximation $\tilde{\mathcal{D}}_\lambda$ of the domain $\mathcal{D}$,*

$$\begin{aligned} & \min f(x) \\ s.t.\ & (W^{z,1}_{k-1} - W^{z,0}_{k-1}) \times z_{k-1} + (W^{z,1}_{k-1} - W^{x,0}_{k-1}) \times x + b^1_{k-1} - b^0_{k-1} \leq \lambda \\ & z_{i+1} = \max(W^z_i \times z_i + W^x_i \times x + b_i, 0) \qquad \textit{for } i = 0, \ldots, k-2 \\ & z_0 = 0 \end{aligned} \tag{25}$$

is equivalent to

$$\begin{aligned} & \min f(x) \\ s.t.\ & (W^{z,1}_{k-1} - W^{z,0}_{k-1}) \times z_{k-1} + (W^{z,1}_{k-1} - W^{x,0}_{k-1}) \times x + b^1_{k-1} - b^0_{k-1} \leq \lambda \\ & z_{i+1} \geq 0 \qquad \textit{for } i = 0, \ldots, k-2 \\ & z_{i+1} \geq W^z_i \times z_i + W^x_i \times x + b_i \qquad \textit{for } i = 0, \ldots, k-2 \\ & z_0 = 0. \end{aligned} \tag{26}$$

Indeed, the domain P is equivalent to the feasible set of problem (25) while Q is equivalent to the feasible set of problem (26). Since the projection of these two sets on the x space is equivalent, *i.e.* $proj_x(P) = proj_x(Q)$, then the feasible set of solutions regarding x and thus the optimal solution x^* of the two optimization problems are the same.

Therefore, $\min_{x \in \tilde{\mathcal{D}}_\lambda} f(x)$ reduces to a linear program if the objective function $f(x)$ is (piecewise) linear (and convex), and if all the activation functions g_i used in the ICNN are piecewise linear, convex, and non-decreasing functions.

C Experiments: considering an objective function values when learning the ICNN

We detail in this section how the training-loss function can be adapted to consider the value of the optimization objective function when training the ICNN classifier, in order to improve the quality of the approximation $\tilde{\mathcal{D}}_0$ in regions close to the optimum. Notice that in order to implement these methods, we assume that for each element i of the training set, we also know the value $f_i = f(x_i)$ of the objective function (in addition to the input x_i and the output y_i indicating constraint satisfaction of x_i w.r.t. $\mathcal{D}$).

In order to force the learnt approximation $\tilde{\mathcal{D}}_0$ to be better in regions where the constrained optimum might be located, we give a higher weight in the loss function to input-output samples with smaller associated values of f. Thus, the loss function for an observation x of class y and corresponding to an objective value f would be given by:

$$\text{loss}(\theta, x, y, f) = w_{y,f} \times \left[-\log \left(\frac{\exp(h_y(\theta, x))}{\exp(h_0(\theta, x)) + \exp(h_1(\theta, x))} \right) \right],$$

where $w_{y,f}$ would depend both on the true class of the sample and on the value of the objective function for this sample.

We tested two methods to compute the weights $w_{y,f}$.

The first method is such that the weights vary linearly between two bounds with the objective function:

$$w_{y,f} = w_y^{\min} + \frac{f_y^{\max} - f}{f_y^{\max} - f_y^{\min}} (w_y^{\max} - w_y^{\min}),$$

where $f_y^{\max}$ and $f_y^{\min}$ are respectively the maximum and minimum values of the objective function among the training samples with label y, and $w_y^{\min}$ and $w_y^{\max}$ for $y \in \{0, 1\}$ are four hyper-parameters that can be tuned.

The other method that we tested is such that the weights $w_{y,f}$ take only two values per class y, a high value when the (x, y, f) tuple corresponds to a "small enough" value of the objective function $f(x)$, and a lower value otherwise:

$$w_{y,f} = \mathbb{1}_{A_y}(f) w_y^{\max} + (1 - \mathbb{1}_{A_y}(f)) w_y^{\min},$$

where $w_y^{\max}$ and $w_y^{\min}$ are the two values the weights of the training samples with label y can take and $\mathbb{1}_{A_y}(\cdot)$ is the indicator function indicating if the sample corresponds to a "small enough" value of f for class y. On can imagine various possibilities to define each one of the two sets A_y so that it focuses on the "small enough" f-values of the corresponding class y.

In the paper, only the results of the second method are shown. In these computations, the set A_0 (of elements of class 0 with "small enough" f) was defined so as to contain the 10% training elements of class 0 ($x \in \mathcal{D}$) showing the smallest values of f; this comes together with a weight $w_0^{\max} = 1$ and $w_0^{\min} = 0.2$. On the other hand, for class 1, we used $w_1^{\max} = w_1^{\min} = 1$ so that the choice of the set A_1 has no impact in this case.