# Continuous Exact Explanations of Neural Networks

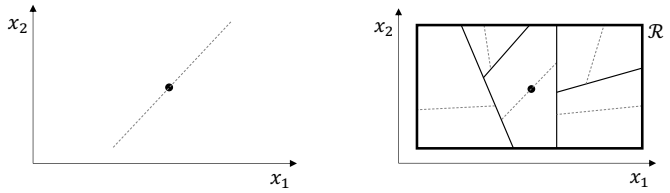
Alice Dethise
*Nokia Bell Labs*

Marco Canini
*KAUST*

***Abstract*—Accurate explanations of how a trained neural network (NN) behaves are desirable for a wide range of labor-intensive activities, including troubleshooting, validation, and understanding performance issues or identifying biases.**

**We address the problem of explaining feedforward piecewise NNs (such as CNNs with RELU) by breaking them down into their linear components. Automatic encoding of the NN structure into linear program constraints is used to extract *continuous exact explanations*, which help interpret model behavior over continuous regions, provide model descriptions that convey the importance of input features and answer model queries that analyze the feature contributions under user-defined constraints.**

**Our examples show that we can extract explanations for NNs with a moderate number of layers without relying on approximations. We demonstrate that the high-level explanations can help understand the outputs of NNs and the comparative importance of features by observing the linear models. We also show how explaining continuous inputs prevents certain attacks that have been proposed against existing explainers.**




**(a) Post-hoc** **(b) Continuous exact (ours)**

**Fig. 1: Illustration contrasting post-hoc explanations versus continuous exact explanations. A post-hoc explanation typically consists of feature weights that yield, as an explanation, a single linear model of the input $\bar{x}$ (shown as a dashed line) approximating the NN around a specific data point. The farther from the data point, the more inaccurate this approximation is. Our approach provides the NN's decision boundaries and the exact linear model for each piecewise-linear subregion of the analysis region. In the common case, for ease of interpretation, this is a 2D bounding box around the data point of interest.**

## I. INTRODUCTION

Explaining neural networks is a significant and necessary step to enhance *trust* in the growing body of ML applications, particularly where the risks of unpredictability and mistakes may bear unacceptable costs. The ML literature has no single, well-accepted definition of what an "explanation" is [1]. However, a flurry of recent works is advancing on the long-standing desire to pierce through the black-box nature of NNs. We aim to take explainability a step further, by proposing an approach helping domain experts through three properties: *completeness, exactness, and explicit validity boundaries*.

We embrace the approach introduced with techniques for post-hoc explanations [2, 3, 4], which offer a way to *locally analyze* a trained NN and determine, for a given input data point, how the model decided on its output, attributing a score or weight to individual input features based on their effects on the output. For data scientists, this kind of local explanation as feature weights provides direct insights into the model behavior and assists in increasing confidence around it or seeking for remedy when the behavior is not the expected one. This approach differs from other works that use a mathematical approach to explore the low-level structure of NNs [5, 6, 7]. Explanations as feature weights yield high-level and human-interpretable explanations, making their interpretation significantly simpler; this is especially useful when the available domain expertise and ML expertise do not overlap.

However, current post-hoc explanations have significant drawbacks. First, they only explain the *specific data points* provided for analysis [2, 8]. With this approach, it is impossible to depict a broader understanding of the model behavior over ranges of continuous inputs. Second, previous works rely on *sampling and approximation* to extract explanations in a reasonable time [9, 10] or build a surrogate model [11, 12]. Approximation introduces several issues: underapproximation can leave errors undetected and creates a vector for attacks [13]; overapproximation can lead to valid models being rejected, sacrificing performance benefits and preventing models from adjusting their behavior for irregular objectives. In summary, prior post-hoc explanations are *too localized and approximate*.

We propose a new explanation method that extracts continuous explanations over ranges of inputs (called the analysis region). Our approach introduces *continuous exact explanations* to divide the model into a set of linear models, from which we extract explanations based on normalized feature weights. Compared to existing techniques, we extract explanations for input subregions rather than single points and do not suffer from approximation. Moreover, we explicitly identify the validity region of explanations (Figure 1).

At its core, our technical approach builds upon methods for the efficient encoding of a feedforward NN as a linear problem, that is tractable using off-the-shelf solvers for many NNs of moderate size and analysis regions. Further, a significant benefit of linear solvers is that they are sound and complete to finite precision.[1] This approach allows us to find every possible set of activation states in the analysis region efficiently without relying on approximations.

Since existing encodings are insufficient to obtain continuous explanations, we contribute a new formalization of contin-

[1]We use $10^{-8}$ precision; errors are essentially negligible.

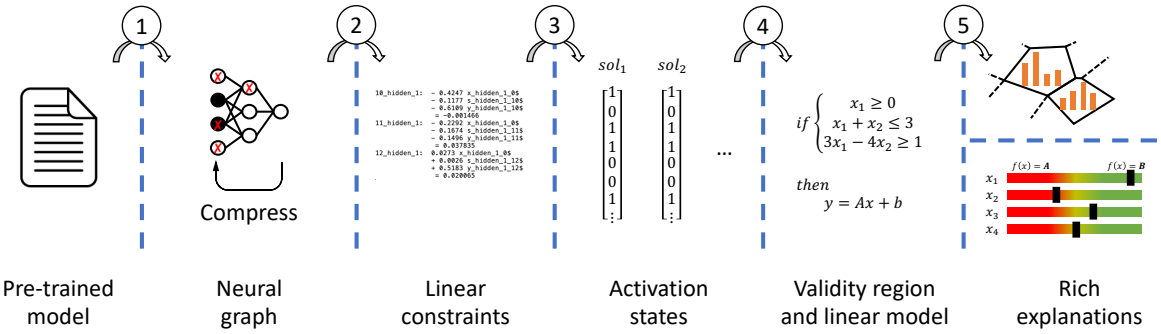



**Fig. 2: Our process to automated continuous exact explanations of NN models.**

uous explanations and an efficient end-to-end approach that (1) exploits an extended encoding to obtain linear models and (2) transforms the linear models into explanations. Specifically, we design an automated way for encoding feedforward NNs and obtaining piecewise linear models that preserve the mapping of inputs to outputs within the identified validity regions (§III).

We express explanations in two forms (§IV): (1) model descriptions encode normalized feature weights, and (2) model queries restrict the explanations to specific model behaviors in terms of its outputs. We also generate a visual representation of explanations that provides meaningful information to data scientists that is not available through extrapolating from single-point explainers (§VI-C).

As with other exact methods [6, 14], we acknowledge that our method fails to scale to large models (e.g., image classification), our work being primarily aimed at such small NNs with intelligible inputs (§V). Further, we make no attempt to address scalability issues via parallelization or accelerators ; therefore, we expect that our method can still be significantly improved in terms of performance and optimizations. However, we already show that this approach is practical for models with a small number of layers; for applications where approximation is acceptable and models using non-feedforward piecewise linear structures, our work can be extended through approximate encodings [15, 16, 17].

We demonstrate the value of our approach through several use case scenarios (§VI). They illustrate how continuous exact explanations help identify which regions of the input space map to a particular output, and show how the weights of input features vary in the input space, confirming that the observed weights match the feature importance of the original model. We also show how adversarial models trained to fool single-point post-hoc explanations can be countered. Finally, we demonstrate with examples how excessive simplification would lead to incorrect interpretations, and why all the information included in the explanations is significant.

## II. Overview

We propose a framework to automatically analyze a given trained NN model and output interpretable explanations for it. Our explanations are without approximation and are valid for continuous ranges of inputs within the desired analysis region ($\mathcal{R}$). We support different explanation types aimed at supporting different use cases, including describing how much each input feature influences the model's outputs and answering detailed queries about the outputs.

The precursor to obtaining explanations is to compute a set of linear models, each of which captures the mapping between inputs and outputs of the NN within a validity region. Figure 2 illustrates the different steps of the process.

① We convert a trained NN into a framework-agnostic neural graph representation. We further compress the neural graph using simple semantics-preserving transformations (e.g., merging successive operations as a single linear transformation and removing identity operations) and optimizations that reduce the number of operations.

② We encode the neural graph as a Mixed Integer Linear Programming (MILP) problem, which defines the relationship between inputs and outputs through a "chain" of variables linked together by constraints. Following [18], we convert the neural graph into variables and we define a set of linear constraints representing the relationship between those variables.

③ Using an off-the-shelf solver (e.g., CPLEX), we find solutions for the MILP. A solution is an assignment of values to each variable, such that those values satisfy the constraints. We obtain one solution for every possible activation state, i.e., the valuation of indicator variables.[2]

④ For each activation state, we determine a linear model along with its corresponding validity region. We leverage the observation that once an activation state is fixed, every neuron of a NN becomes a linear transformation of its inputs. Thus, we extract a set of functions of the form $y = a_0 + a_1 x_1 + \ldots + a_m x_m$. As the NN is piecewise linear, each function expresses the NN in a certain region of the inputs $x$. We then determine the constraints on the inputs $x$ that yield the validity region of each linear model.

⑤ We extract high-level explanations for the analysis region $\mathcal{R}$. We define two types of explanations. 1) A *model description* provides an importance score for each input towards the output, where a higher score means that a small change in the input value leads to a larger shift in the output value. 2) A *model query* is a search within $\mathcal{R}$ for either the subregion where a model's output has the highest value among all outputs or the subregion where the model attributes a higher value to one output over another.

[2]Although the number of possible states is exponential in the number of non-linear activations, NNs are often overparameterized and thus, the number of realizable activation states is in practice much smaller than the total one.

## III. NNs As Linear Transformations

We formalize the definition of exact (without approximation) and complete (explaining every point within continuous bounds) linear transformations for piecewise linear models and review their encoding. We then describe how to extract those transformations for *feedforward piecewise linear NNs*.

### A. Defining the model as linear transformations

Let $f : \mathbb{R}^m \to \mathbb{R}^n$, a piecewise linear function, be the original prediction model to be explained. Without introducing any approximation or inaccuracy, we seek to obtain $g^{\mathcal{R}}(x)$, a simplified representation of $f(x)$ whenever the model inputs $x$ is in the analysis region $\mathcal{R}$. Let $g : \mathbb{R}^m \to \mathbb{R}^n$ denote guarded affine transformations, the composition of affine transformations guarded by Boolean predicates. An affine transformation $h$ is in the form $h(x) = \hat{\mathbf{w}} \cdot x + \hat{b}$, with $\hat{\mathbf{w}} \in \mathbb{R}^{m \times n}$ and $\hat{b} \in \mathbb{R}^n$. The guard expression is a function $e : \mathbb{R}^m \to \{\mathit{True}, \mathit{False}\}$ as a conjunction of linear constraints: $e(x) := \bigwedge_i (c_i(x) \geq 0)$, where $c_i(x)$ is a linear function.

**Definition 1** (Piecewise linear representation)**.** Let $h^1, \ldots, h^k$ be a set of affine transformations that are equivalent to $f$ *for some* $x$, and $e^1, \ldots, e^k$ a set of guard expressions, such that every value of $x \in \mathcal{R}$ satisfies exactly one of the guard expressions. We define $g^{\mathcal{R}}(x) := \{h^k(x) \mid e^k(x)\}$, such that $\forall x \in \mathcal{R} : \exists k \mid e^k(x)$.

Note that there exists a mapping $e^k \to A^k$ defined as $A^k := \{x \mid e^k(x)\}$. Thus, the constraint on $\forall x \in \mathcal{R} : \exists k \mid e^k(x)$ is equivalent to $\cup A^k = \mathcal{R}$. $A^k$ is the validity region of $h^k$.

### B. MILP encoding

We briefly review our encoding, which derives from the one by [18]. We remark that, although this encoding is not novel, we introduce several non-trivial extensions in §IV.

The encoding follows the layer-wise composition of the NN. Consider layer $l \in [1, L]$ with inputs $x^l \in \mathbb{R}^{n^{l-1}}$ ($n^0 = m$) and outputs $y^l \in \mathbb{R}^{n^l}$. The model parameters (weights $W^l$ and biases $b^l$) are treated as constants. When not needed, we omit the superscript $l$ to simplify the notation. The layer's relation $y = \text{ReLU}(Wx + b)$ yields as variables: $x$, $y$, the ReLU state indicator variables $z \in \{0, 1\}^n$ and auxiliary variables $s \in \mathbb{R}^n$. The constraints below are used:

$$\forall i \in \{1, \ldots, n\}, \quad y_i = \textstyle\sum_j W_{i,j} x_j + b_i - s_i;$$
$$z_i = 1 \Rightarrow s_i = 0; \quad z_i = 0 \Rightarrow y_i = 0.$$

That is, $z_i = 1$ denotes an activated ReLU, which propagates its input value ($x \geq 0$); otherwise, $z_i$ is 0. The auxiliary variable $s_i$ captures the residual value when the constraint $y_i = 0$ holds. Encoding layer $(l+1)$ introduces the propagation constraint $x^{l+1} = y^l$, and enables the solver to propagate the inputs $x^1 \in \mathcal{R}$, to the outpus $y^L$ in a NN with $L$ layers.

### C. Extracting linear models

We feed the encoding to an off-the-shelf solver to compute all possible activation states in $\mathcal{R}$, representing the possible states of all ReLUs in the NN (*on* if they propagate their input value, *off* otherwise). Finding all solutions is a hard problem that can potentially take a long time (and dominates the time to create explanations); however, by leveraging existing optimizations, the time remains practical in many examples.

We extract the linear models $h^k(x) = \{h_1^k(x), \ldots, h_n^k(x)\}$ for each of the $n$ output neurons. $k$ indexes the activation states. For every activation state $k$, we resolve the non-linearity introduced by ReLU neurons by substituting their output according to the value of the corresponding indicator variable $z$ at each layer. This reduces the value of each layer's output in the NN to a recursive linearization function $y^l = v^l(x; W^l, b^l)$, where $v^l$ is a linear function with $i$-th component in the form $v_i^1(x) = x_i$ and for $l > 1$:

$$v_i^l(x) = \begin{cases} 0, & \text{if } z_i^l = 0 \\ \sum_{j=1}^{n^{l-1}} W_{i,j}^l v_j^{l-1}(x) + b_i^l, & \text{if } z_i^l = 1 \end{cases} \quad (1)$$

We simplify the value of each neuron as a linear transformation in the form $v_i^l(x) = \hat{\mathbf{w}}_i^l \cdot x + \hat{b}_i^l$, with $\hat{\mathbf{w}}_i^l \in \mathbb{R}^m$ and $\hat{b}_i^l \in \mathbb{R}$. The linearized parameters are equal to:

$$\hat{\mathbf{w}}_i^l = \begin{cases} 0, & \text{if } z_i^l = 0 \\ \sum_{j=1}^{n^{l-1}} W_{i,j}^l \hat{\mathbf{w}}_j^{l-1}, & \text{if } z_i^l = 1 \end{cases} \quad (2)$$

$$\hat{b}_i^l = \begin{cases} 0, & \text{if } z_i^l = 0 \\ \sum_{j=1}^{n^{l-1}} W_{i,j}^l \hat{b}_j^{l-1} + b_i^l, & \text{if } z_i^l = 1 \end{cases} \quad (3)$$

where $\hat{\mathbf{w}}_i^1$ is a one-hot vector with 1 at $i$ and $\hat{b}^1 = 0$. We then collect the value function $v_i^L(x)$ of each output neuron $y_i$ as the linear transformation $y_i = \hat{\mathbf{w}}_i^L \cdot x + \hat{b}_i^L$ (with $L$ being the output layer). Then, $h_i^k(x) = y_i \equiv v_i^L(x)$ is an affine transformation and satisfies definition 1. This technique can also provide the value function for intermediate layers of the network, although there is no guarantee that such information would be meaningful because the values in intermediate layers may not be interpretable.

### D. Extracting validity regions

Recall from §III-A that the validity region $A^k$ of $h^k$ is equivalent to the guard expression $e^k(x)$. To extract $e^k(x)$, we note that for each of the $k$ activations states $z = z_1, \ldots, z_N$, the activation conditions of the ReLU function yield a sequence of constraints $c_i^l(x)$ of the form:

$$c_i^l(x) = \begin{cases} z_i^l = 1 \Rightarrow & \sum_j W_{i,j}^l x_j^l + b_i^l \geq 0 \\ z_i^l = 0 \Rightarrow & \sum_j W_{i,j}^l x_j^l + b_i^l \leq 0 \end{cases} \quad (4)$$

By substituting for $x_j^l$ the linearized representation of the neuron computed in §III-C, we can express the value of the layer inputs as a linear function of the NN inputs, $x_j^l = y_j^{l-1} = \hat{\mathbf{w}}_i^{l-1} \cdot x + \hat{b}_i^{l-1}$. Thus, $c_i^l(x)$ is a linear constraint on $x$ and satisfies the conditions from §III-A. Since the activation states $z$ returned by the solver are valid for the analysis region $\mathcal{R}$, we express $e^k(x)$ as the following conjunction: $(\bigwedge_{l,i} c_i^l(x)) \cap \mathcal{R}$. As $e^k(x)$ is the intersection of linear constraints on $x$, it must necessarily yield a convex polytope $A^k$ in the analysis region. We can then simplify the set of constraints by computing the convex hull of $A^k$ to express it through the vertices of the polytope. This removes redundant constraints and yields the minimal set required to express $e^k$.

The MILP encoding blocks each previously-found set of activations $z$, to ensure that no two solutions are identical.

As described above, each validity region $A^k$ is associated with a single $z$ and the number of (non-empty) regions is finite, which guarantees a finite number of solutions even for continuous inputs. Figure 1b illustrates in 2D how a NN is broken down into its linear components. Each area represents one validity region, and is associated with a linear function of the inputs. In general, the validity regions $\{A^k\}$ are $m$-dimensional polytopes (for $m$ input features), rather than 2D.

### E. Generalizing to any piecewise linear layer

The approach discussed in §III-C describes the extraction of linear constraints for a fully-connected layer with RELU activations. We can generalize this approach to any kind of linear layer. For instance, a convolution layer is encoded similarly to a fully-connected layer, but the weight matrix is replaced with the values of the convolution filter, while the inputs are replaced with the convolution window.

Given an arbitrary piecewise linear layer $l$ with input $x^l = \hat{\mathbf{w}}^{l-1} \cdot x + \hat{b}^{l-1}$, and its MILP encoding using indicator variables $z^l$, we define the operation of this layer by the following transformation: $\Phi^l(\hat{\mathbf{w}}^{l-1}, \hat{b}^{l-1}, z^l) \mapsto \hat{\mathbf{w}}^l, \hat{b}^l, c^l$. This transformation propagates the weights from layer $l$ to layer $l+1$, enabling extracting the linear models from NNs with any piecewise-linear layer, including convolution and fully-connected layers, Leaky RELU or Parametric RELU activations. This also allows the automatic encoding of typical NN transformations like slicing and reshaping.

## IV. MODEL EXPLANATIONS

The obtained coefficients $\hat{\mathbf{w}}^L$ of the linear model $h^k$ and validity region $A^k$ describe the model, but lack ease of interpretation. Their usefulness is also reduced when NNs are used as a tool for nonlinear approximation because they do not reveal when a subregion (i.e., multiple contingent validity regions) exhibits linear or quasi-linear behavior. Treating the model as entirely linear also carries risks of misinterpretation. Thus, we avoid using directly the linear models, but instead focus on *feature weights*, as in [2, 3, 4].

Although it is possible to convert the low-level information extracted in §III during post-processing, doing so comes at an additional cost of higher complexity and lower performance. Thus, in this section, we develop two critical aspects: $(i)$ we formally define the relation between low-level and high-level, human-interpretable explanations, and $(ii)$ we express that relation as constraints in the MILP encoding, enabling our approach to produce high-level explanations from the solver and with minimal additional processing.

As we make no assumptions about the expected input ranges, the above coefficients are not proportional to features' importance; they simply reflect how an $\epsilon$ variation in input values reflects on the output, despite that the features might have different scales. Thus, we seek to normalize those coefficients to obtain a set of weights that satisfies the following ranking criterion.

**Definition 2** (Ranking Criterion)**.** The feature with the $k$-largest weight provides the $k$-largest change in the output when the feature value varies in its expected range.

### A. Model description

Given the linear model $h^k$ with output vector $h^k(x) = \{h_1^k(x), \ldots, h_n^k(x)\}$ holding the value of all output neurons, we treat each component of the output as a different linear model and seek to extract the weights for all of them separately. In the rest of this section, we assume that we are explaining a single component $h_o^k(x) = w_1x_1 + \ldots + w_mx_m + c_o$, and determine the set of parameters $w_1, \ldots, w_m, c_o$ of each input feature for that component. We first show how the model coefficients can be normalized assuming, for simplicity, that each feature has uniform distribution and bounded range. Later, we generalize the weights to the unbounded-range case and non-uniform distributions.

Assuming the valid range of inputs is a $m$-dimensional box in which each feature $x_i$ can vary in the range $[a_i, b_i]$, the linear function can be written as $h_o^k(x) = \sum_{i=1}^m w_ix_i + c_o,\ a_i \leq x_i \leq b_i$. To express weights independent of the input range, we express the equivalent model $\bar{h}_o^k(\bar{x}) = \bar{w}_1\bar{x}_1 + \ldots + \bar{w}_m\bar{x}_m + c_o$ where each component of the input $\bar{x}_i$ has range $[0, 1]$. We write the model as $\bar{h}_o^k(\bar{x}) = \sum_{i=1}^m (\bar{w}_i\bar{x}_i + c_i) + \bar{c},\ 0 \leq \bar{x}_i \leq 1$ with $\bar{c} = c_o - \sum_{i=1}^m c_i$.

To ensure that the models are equivalent on ranges $[a_i, b_i]$ and $[0, 1]$, we require a mapping of $x_i$ to $\bar{x}_i$ such that $h_o^k(x)|_{a_i}^{b_i} \equiv \bar{h}_o^k(\bar{x})|_0^1$. This equivalence is guaranteed by selecting $c_i, \bar{w}_i$ such that:

$$w_ix_i|_{a_i}^{b_i} = (\bar{w}_ix_i + c_i)|_0^1 \qquad c_i = w_ia_i \qquad \bar{w}_i = w_i(b_i - a_i)$$

We conclude that the weight of each variable $x_i$ is $\bar{w}_i = w_i(b_i - a_i)$. This value is equal to the variation of the output of $h_o^k(x)$ when $x_i$ varies between the upper and lower bound. This behavior satisfies definition 2, and thus *the final explanation uses the weights $\bar{w}_i$ representing the importance of feature $x_i$.*

The weights used above assume that $x_i$ is uniformly distributed in the range $[a_i, b_i]$. In practice, many problems use inputs following a non-uniform distribution in an unbounded or semi-bounded range. For the remainder of this subsection, we assume that the distribution of $x$ is unknown, but we have samples of it (such as data points from the testing set). We can assume that given enough samples, the mean $\bar{X}_i$ of the samples of $x_i$ follows the normal distribution $N(\mu_i, \sigma_i^2)$. We know that $\bar{X}_i$ lies in $[\mu_i - k\sigma_i, \mu_i + k\sigma_i]$ with very high probability for $k \geq 3$. Thus, we compute the weight without outliers by using the probabilistic bounds $a_i = \mu_i - k\sigma_i$ and $b_i = \mu_i + k\sigma_i$. The normalized weights are $\bar{w}_i = w_i(b_i - a_i) = w_i(2k\sigma_i)$. The coefficient $2k$ is shared between all features and can be ignored without loss of accuracy. The final weights are $\bar{w}_i = w_i\sigma_i$.

### B. Model queries

We define a model query as a specific type of explanation for which the analysis region $\mathcal{R}$ is dependent upon specific query parameters, such as constraints on input values, constraints on output values, and comparison between the model outputs. Restricting the analysis region solely based on input values is straightforward to encode in the MILP search. Therefore, we focus on cases where the analysis region depends on some value of the model output.

**Finding highest value outputs.** A typical use case for output constraints is explaining classifiers. It is common to encode the $n$-class decision as $n$ output neurons in the last layer such that the neuron with the highest value is selected as the decision; thus, generating explanations for a region where a particular class is selected requires expressing constraints between the output neurons. As such, we seek to satisfy a model query in which each guard expression is tied to a pair (linear transformation, model output), which restricts the analysis to regions where the analyzed output is highest.

To search for query explanations, we extend the MILP encoding with additional auxiliary variables $r_{ab}$ and $R_a$ to express the rank of each output neuron, for the top class $a$ and any $b \in [1, n]$. Given $n$ output neurons with linear models $h_1^k, \ldots, h_n^k$, the auxiliary variables are encoded with the constraints:

$$r_{ab} = 1 \Leftrightarrow h_a^k(x) \geq h_b^k(x) \qquad R_a = 1 \Leftrightarrow \textstyle\sum_{i=1}^{n} r_{ai} = n \quad (5)$$

Recall from §III-D that we use the indicator variables to "block" solutions and guarantee that the solver returns exactly one solution per valid assignment of activation states $z^k$. By adding $R_a$ to the set of *blocking variables*, the solver returns solutions for the pair of ($z^k$, $a$). The soundness of the solver guarantees that for each extracted solution, there exists some input value $x$ that generates the activation state $z^k$ and for which $h_a^k(x)$ has the highest value among all outputs. We restrict the validity region extracted in §III-D so that $A^k$ contains only the values $x$ for which $h_a^k(x)$ has the highest value, i.e., $h_a^k(x) \geq h_i^k(x)$ for $i \in \{1, n\} \backslash a$. This yields $n-1$ constraints, which are added to the previously extracted guard expression $e^k(x)$: $e_a^k(x) = \wedge_{i=1, i \neq a}^{n} (h_a^k(x) - h_i^k(x) \geq 0) \wedge e^k(x)$.

**Comparison between outputs.** Another type of query is to extract, given an output $a$ with the highest value, the weights representing how much each feature "favors" this output against the others, such that *the feature with the largest weight is the one which, for a positive change, will most increase the gap between the highest output and the others*. In contrast, the feature with the lowest weight is the one for which positive changes will *decrease* said gap.

We compute the weights as in §IV-A, using as linear model $h_{ab}^k(x) = h_a^k(x) - h_b^k(x)$. We tie the guard expression for this model to a triplet (linear transformation, highest output, second-highest output).

The MILP encoding is extended through another auxiliary variable $R'_b$ which is defined as $R'_b = 1 \Leftrightarrow \sum_{i=0}^{n} r_{bi} = n - 1$, i.e., $R'_b$ is one only when $h_b^k(x)$ is the *second* highest output. We can extend the computation of activation states presented above to find all triplets ($z^k$, $a$, $b$) by including both $R_a$ and $R'_b$ to the set of blocking variables. We extend the guard expression such that:

$$e_{a,b}^k(x) = \wedge_{i=0, i \notin \{a,b\}}^{n} (h_b^k(x) - h_i^k(x) \geq 0) \wedge e_a^k(x) \quad (6)$$

Each area will automatically satisfy the constraint that $a$ is the highest output and $b$ is the second-highest. To return the explanation for top class $a$ and *any* second-highest output, we simply run the solver once for each possible value of $b$.

*1) Softmax normalization of outputs:* The approach for comparing model outputs described in §IV-B assumes that every layer of the NN is piecewise linear. In practice, classifiers commonly use a softmax function to normalize the values of the last layer. However, the softmax function $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$ is non-linear, thus a NN using softmax normalization is not a piecewise linear function and is not a composition of affine transformations.

Nonetheless, if the softmax function is only used in the model's last layer, it is still possible to generate model queries. We observe that $\sigma(\cdot)$ is strictly increasing, meaning that we can use the property $x > y \Rightarrow \sigma(x) > \sigma(y)$ to find which of the output neurons has the highest value. Thus, we encode the value of each neuron in the layer immediately before the softmax normalization, which is a linear function, and use them to order all outputs according to their value using the method described before.

### C. Use cases

Our continuous exact explanations as introduced above go beyond traditional post-hoc explanations: they support a range of NN troubleshooting tasks for continuous regions of the input space instead of individual data points (c.f. Figure 1). We depict below several use cases that are interesting based on our own experience.

**Mapping the model outputs over continuous regions.** Continuous exact explanations can map regions of the input space to a particular behavior of the model output. For example, we can find the exact set of all inputs that lead to a particular decision being selected for classification models.

**Detecting irregular behavior.** We can identify when the behavior of the model is changing rapidly in a region, for example, because very small changes in the inputs lead to large and unexpected changes in the outputs. Conversely, we can also identify if the model has wide regions where the output shape is smooth.

**Identifying the most important features.** Explanations can reveal which features are most important for a particular output to occur. This helps validate that the model uses information known to be important, or does not use some protected feature.

**Predicting direction of changes.** For models with multiple outputs, in addition to feature importance, we can also determine the direction in which the output will move. An example is determining, for a classifier, which output is most favored by a variation in some input feature. We can also identify the tipping point where an increase in feature value stops being favorable to a particular choice and becomes detrimental.

## V. Discussion

### A. Scope of applicability

Our approach guarantees the completeness and exactness of explanations. This comes at the expense of general applicability for large NNs, due to the limited scalability of MILP encoding. We also assume that features are intelligible, a necessary property for post-hoc explanations to be useful. These characteristics remain common for practical applications of NNs for classification or decision-making tasks: e.g., [19, 20, 21, 22] use between 2 and 12 layers and intelligible features.

Our approach is not applicable to computer vision or language models. And while our approach aids in troubleshooting, it serves the needs of explainability, which is orthogonal to verification and certification of NNs [9, 23].

The time to generate explanations is mainly constrained by the MILP size and the performance of the solver. By using an off-the-shelf solver, we inherit all of its advanced optimizations; but there is no upper bound on solving time [24]. This issue can only be overcome through approximation, which goes contrary to the purpose of this paper. All examples NNs provided in this paper take between seconds to a few minutes to solve, which we believe attests to the practicality of our approach in the analyzed case studies (§VI). Our examples show that models with similar complexity significantly vary in explanation time, and the model size is not a good indicator.

### B. Analysis regions

The analysis region $\mathcal{R}$ can be any subregion of the input space, in which the solver will generate explanations. The user defines this region by specifying a set of $i$ linear constraints on the input $x \in \mathbb{R}^m$ where $x_j$ is the $j$-th input feature as follows: $\mathcal{R} := \bigwedge_i (c_i(x) \geq 0)$, with $c_i(x)$ in the form: $a_0 + a_1 x_1 + \ldots + a_m x_m$. As discussed in §IV-B, $\mathcal{R}$ can be used, for instance, to encode queries with input constraints or to indicate the ranges of input features.

We present strategies for selecting the analysis region of an explanation, either to accelerate the process to meet time constraints, or because the search space is too large and impractical for interpretability. We use these strategies in §VI.

**2D representations.** An explanation's validity region is always a non-empty subset of the analysis region; generally, the dimensions of the validity region and the analysis region are equal. At the same time, ML models typically have many input features, representing high-dimensional inputs. This creates a challenge where the extracted high-dimensional explanations cannot be accurately represented visually or easily understood by data scientists, reducing the use of extracted information. To mitigate this issue, a strategy is to restrict the analysis region to only two input variables, representing a cut in two dimensions of the input space. This yields explanations that can be represented visually and helps with interpreting correlations between any 2 features and model's outputs.

**Cloud of points.** The points of interest for model explanations do not always fill the entire input space uniformly. For example, the model could receive many queries generated by end-users for values concentrated around some hotspot, or a monitoring process might have identified points in which the model selected an undesirable output. Generating explanations in that region can both explain the model behavior, and detect new, undiscovered points of interest in the vicinity. We achieve this by defining the analysis region as a convex envelope or bounded hypercube containing all points in the cloud.

**Individual points.** While we offer the ability to explain every instance in continuous regions, some queries might not need this versatility and only require information about individual points. In this case, continuous exact explanations still have several benefits over single-point post-hoc explanations: they guarantee that the extracted model represents the actual weights of the model for those points without approximations, and can provide the validity region for that linear model.

## VI. Case Studies

The advantages of continuous exact explanations over existing post-hoc explanations techniques are two-fold. First, owing to their exactness and completeness properties, they inherently improve confidence in post-hoc explanations for single data points (c.f. Figure 1). Second, thanks to their continuous nature, they open up new possibilities in understanding a NN's behavior by analyzing the function learned by the NN.

Ultimately, the most tangible benefits of explainable NNs lie in the hands of the users, who will be able to use the explanations to make informed decisions on the trustworthiness of the models, use domain expertise to find and correct flaws, or learn about the system and data features by applying human understanding to the behavior that the model learned.

We now present various case studies to showcase some of the possibilities of how humans can discover new information through continuous exact explanations via visual inspection.

### A. Which inputs lead to a particular output?

A main activity involved in interpreting or troubleshooting NNs is to make sense of how the inputs are mapped to their outputs. This question is one of the most basic and informative that one can ask, and yet finding the answer is challenging. Evaluating the NN on a specific input reveals the output, yet, due to the nonlinearity of NNs, we remain oblivious to how even slightly different inputs will be handled. Using post-hoc explanations is impractical: it would require many input samples to attempt to scrutinize the NN's decision boundaries, and the results would anyway remain an approximation without guarantees of accuracy. Similarly, other techniques, which attempt to answer this question directly, either cannot guarantee an answer for every point in the analysis region (lack of completeness) or overapproximate the output (lack of exactness) [23, 25].

Answering this question using continuous exact explanations, however, is straightforward: using model queries to generate explanations only for desired outputs yields the mapping of the outputs for the entire input space.

Figure 3 shows a visual representation of this analysis for Pensieve [19], a nontrivial multi-class decision-making problem for bitrate selection in adaptive video streaming applications. This model is trained to select the optimal video bit rate in the context of adaptive video streaming. The model takes 25 inputs: the previously selected bit rate, the measured network throughput (over 8 past decisions), the measured network latency (over 8 past decisions), the duration of video stored in the buffer, the number of remaining video chunks, and the size of available chunks for the next decision (for each of the 6 bitrates).

The Pensieve NN has 6 outputs, each representing one out of 6 bitrates to choose from: 300, 750, 1200, 1850, 2850, 4300

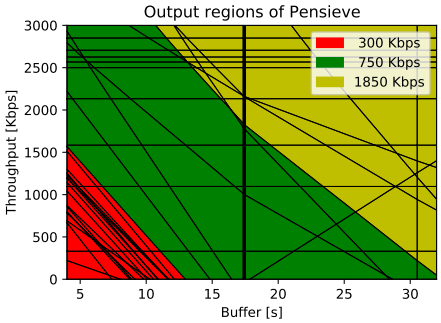



**Fig. 3: Bitrate decision for Pensieve over a continuous range for the two most important features.**

Kbps. The model is trained to maximize the video quality while minimizing the rebuffering time and the changes in video quality. This model contains two hidden layers. The first one is a convolution layer with 768 nodes; the second is a fully-connected layer with 128 nodes. Both hidden layers use RELU activations. The output layer contains 6 nodes with Softmax normalization (the node with the highest value is selected as the chosen bit rate).

We select a 2D analysis region by using the constraints $x_i = K_i$ for some constant value $K_i$ (we use the dataset average of each feature), for all but two features. In this case, we simplify the explanation to 2D for easy visualization, but this could also be used to find all points mapping to some output in higher dimensions.

The results are shown in figure 3. Each of the areas (bordered by black lines) shows the validity region for one linear model $A^k$; the color represents the highest-valued output neuron in that area. The red area shows the output $X_{300Kbps}$, the green area shows $X_{750Kbps}$ and the yellow area shows $X_{1850Kbps}$. From this figure, we can visually confirm that the orientation of the decision boundary matches our expectations: the selected bitrate will increase when the amount of data in the buffer or the achieved throughput increase. We also observe that the boundary is monotonous and there is no point where the conditions are more favorable but the selected bitrate is lower. Finally, while Pensieve has six possible outputs (300, 750, 1200, 1850, 2850, and 4300 Kbps), we can also see that the model transitions from 750Kbps to 1850Kbps and that there is no input between the two regions where the model would select 1200Kbps.

Related questions can be answered with this type of explanation: see whether the boundary is smooth when transitioning from one output to the next, check that there is no enclaved region where one output is abnormal compared to surrounding ones, or find whether a particular output is ever emitted.

### *B. Validating NN against prior expectations*

In the presence of pre-existing information about how the model should behave, such as a collection of test points and domain expertise, we want to check if the model's behavior matches these prior expectations. This leads to the question: is the NN able to capture the target function and does it weigh features based on their (presumed) relative importance?

A domain expert could use post-hoc explanations to extract the weights of features for known points [2, 3]. However, explaining the model in this way is limited to local validation: it remains impossible to know if the behavior is also consistent in the broad vicinity of those points. Another possible approach would be to use verifiers, rather than explainers [9, 10, 26]. This approach assumes that *all* the expected behavior can and will be expressed as verifiable properties, which is impractical and incompatible with experts' intuition. Rather than a replacement, explanations may complement verification and provide an avenue for discovering new properties.

Continuous exact explanations not only help validate the model behavior in a continuous region chosen by the domain expert, but also help answer new questions, such as how the importance of a feature in one region varies compared to a neighboring region, or how broad is the area for which a particular behavior was learned.

To show how we can answer this type of question, we trained a NN with a target *Piecewise* function for which the feature importance is known (in each quadrant, only a subset of the inputs is used). For the use case in section VI-B, we trained a network with a known, piecewise function, which is divided into multiple areas such that each area only uses a subset of the inputs and ignores the rest. We then extract the continuous exact explanations for the trained model and compare them to the parameters of the original function. The approximator is a NN with two fully-connected hidden layers of 32 neurons each, with RELU activations. The NN takes five parameters in the interval $[0, 1]$ as input and returns an estimation of the function value. We train the network with a piecewise function $f : [0, 1]^5 \rightarrow [0, 1]$ defined as

$$
\begin{aligned}
f(x_1, x_2, x_3, x_4, x_5) = \\
&x_1 < 0.5 \wedge x_2 < 0.5 \Rightarrow |cos(\pi \cdot (x_1 + x_3))| \\
&x_1 \geq 0.5 \wedge x_2 < 0.5 \Rightarrow (e^{x_2 \cdot x_4} - 1)/(e - 1) \\
&x_1 < 0.5 \wedge x_2 \geq 0.5 \Rightarrow x_5 \\
&x_1 \geq 0.5 \wedge x_2 \geq 0.5 \Rightarrow (x_2 + x_3 + x_4)/3
\end{aligned}
$$

Each quadrant uses only a subset of the inputs (Table I); all other features are ignored (have no correlation on the output).

We then compare the true function with the one that was learned by the NN, ranking the input features according to their weights, shown in Figure 4. For the analysis region of this example, we set the range of $(x_1, x_2)$ as the full original range $[0, 1]$ and set the other variables to $x_3 = x_4 = x_5 = 0.5$.

For each explanation, we extract the weights of all features, then assign ranks such that the feature with the highest weight has the lowest rank (because the weight are continuous and this is not a classification problem, the weights are never identical). Assuming perfect accuracy (that is, if the NN has perfectly captured the importance of features in the original function), for an input in a quadrant where $m^s$ features are significant, the rank of each relevant feature will be $\leq m^s$ and the rank of ignored features will be $> m^s$.

| Quadrant | $x_1$* | $x_2$* | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|
| Top-left | | | | | ✓ |
| Top-right | | ✓ | ✓ | ✓ | |
| Bottom-left | ✓ | | ✓ | | |
| Bottom-right | | ✓ | | ✓ | |

**TABLE I: Significant features in each quadrant of the function.**

As we can see in Figure 4, the rank of features *generally* matches the ones used in the training function. However, this behavior is not captured perfectly: in several subregions (some of which have been marked with blue stars), a feature that does not correlate with the output is assigned a higher weight than one that does, i.e., the function learned by the NN for those points does not match the training data; this behavior is to be expected without specialized algorithms. Detecting such instances is critical in improving or retraining the model.

### C. Comparison to LIME and SHAP

Continuous exact explanations can explain large areas, but potentially take time to do so. Thus, in applications where exactness is optional, we question whether it would be more efficient to simply run single-point post-hoc explainers multiple times. We apply this technique to compare continuous exact explanations against LIME [2] and SHAP [3]. We stress that it is difficult to quantify the cost of losing exactness, which can vary significantly between different models.

*1) Points as approximation in a continuous region:* We first contrast the visual representation of weights of a feature in a continuous region versus repeated sampling. Section VI-B describes how continuous explanations can produce a visual representation of the importance of features in a decision; we approximate the same result through single-point explanations by sampling multiple random points in the search space, as shown in Figure 5. The number of samples is tuned so that LIME and SHAP use the **same amount of time** as continuous exact explanation.

The number of sample points for single-point explainers is not fixed; instead, we use the time required to extract explanations with the continuous exact method as the time budget, then repeatedly sample the visualization space for explanation points. Such a method depends on the model being explained: a model that is too complex will take longer to compute continuous explanations, whereas single-point explanations will reach a ceiling in the clarity of their explanation with a lower time budget; while a model that is too simple will overly bias the result in favor of continuous explanations.

*2) Prediction stability:* Methods relying on sampling will inevitably have variance in their output when run multiple times; thus, we compare in Figure 6 the results obtained by extracting continuous exact explanations for a single point versus running LIME and SHAP multiple times. Error is measured as the squared distance to the mean importance score found by the respective method. Continuous exact explanations do not have any variance and always return the exact weights.

*Because their value defines the quadrant, $x_1$ and $x_2$ correlate with the output near the quadrant boundaries.

### D. Explanations in higher dimensions

While visual feedback is easier to obtain in two dimensions, it is insufficient to understand the behavior of a NN that has a larger number of features, as only two can clearly be shown on the axes. An alternative solution is shown in Figure 7, in which we can represent the analysis region according to only two features, while extracting for each validity region (four of them being shown) the weight of all 11 features.

### E. Aiding with feature selection

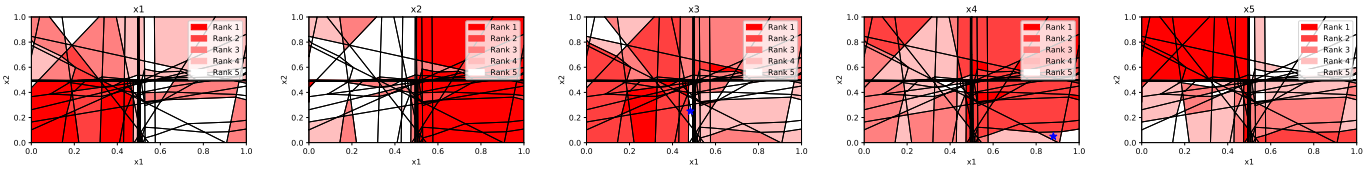
Training an ML model typically requires feature engineering, transforming input features to improve the performance and reduce the complexity of the model. Although such a step is not always necessary for NN models, it nonetheless provides better performance and more cost-efficient models [27]. One common approach is feature selection, in which the input features that do not affect the output are discarded. This is typically achieved by observing the correlation between the features and the desired output.

However, as shown in §VI-B, the function learned by the NN does not always perfectly capture trends in the underlying data (this can be especially true in NNs with few layers). Using explanations, it becomes possible to select the features that are most useful *for the NN* in reaching its decision, rather than from the data. To use this approach, it is important to correctly estimate the importance of features; because the weights computed by single point methods are not representative of the neighboring region, they are unreliable to judge the global feature importance for the NN.

On the other hand, continuous explanations reveal not only the weight of features but also the area for which the explanation is valid. We use this property to compute the total weight of features, defined as the sum of weights in each explanation scaled proportionally to the volume of the validity region. We demonstrate this idea by training two NN classification models on the Credit [28] and Wine Quality [29] datasets under a varying number of features.

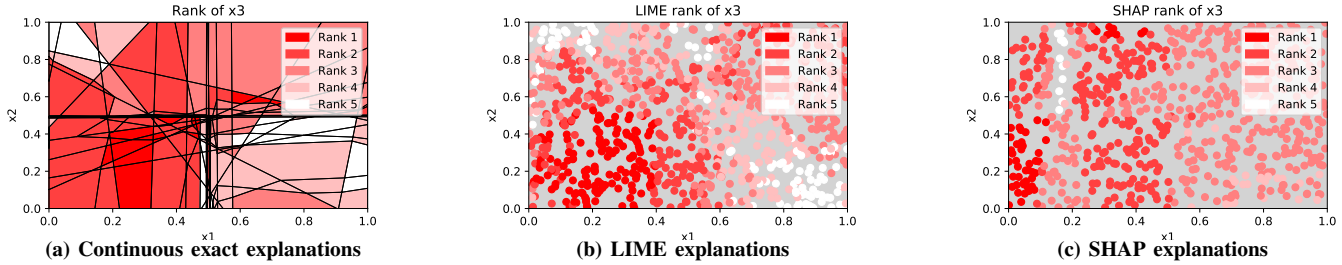
Note that we report the model accuracy *for the entire dataset distribution*, even though the weights are only computed in the analysis region. This approach guarantees that we fairly report the loss in model accuracy, but might not be reflective of real use-case scenarios (i.e., the accuracy of the model *in the analysis region* might be the metric of interest for some applications). We scale the weights for each explanation according to the size of their validity region; i.e., given the validity region $A^k$ and an explanation with weights $\{w_0^k, \ldots, w_n^k\}$, the total weight of feature $i$ is $\frac{\sum_k V(A^k) \cdot w_i^k}{\sum_k V(A^k)}$, where $V(A^k)$ is the volume of the validity region.

Column "All" shows the model accuracy when all the input features of the original model are used. When used three selection criteria to reduce the number of features used: the first criteria is to select a fixed number of features equal to half of the original features (rounded up); the second and third criteria only select features whose weight is greater than 10% (resp. 50%) of the highest weight among all features.




(a) Rank of $x_1$ (b) Rank of $x_2$ (c) Rank of $x_3$ (d) Rank of $x_4$ (e) Rank of $x_5$

**Fig. 4: Weight of individual features (here $x_3$ and $x_4$) in the model, ranked against the remaining features. Regions of a darker color show where the explained feature is more important than the others.**




(a) Continuous exact explanations (b) LIME explanations (c) SHAP explanations

**Fig. 5: Importance of feature $x_3$ in the piecewise model, ranked against the remaining features, using continuous explanations and single-point approximations. Explanations were computed with equal time budget.**

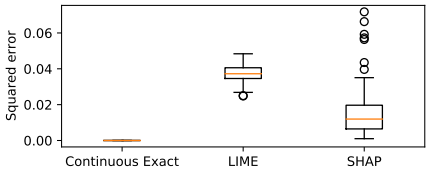



**Fig. 6: Total squared error when the *same point* is explained 200 times.**

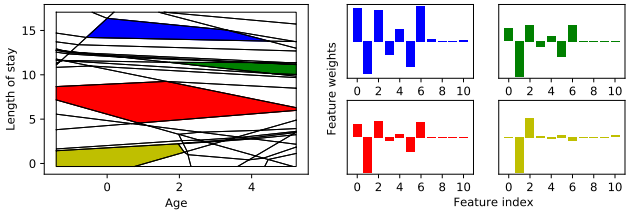



**Fig. 7: Left: 2D validity regions of the adversarial COMPAS model. Right: Weights of all 11 features in marked regions.**

| Model | | All features | Top 50% | $w \geq 0.1\hat{w}$ | $w \geq 0.5\hat{w}$ |
|---|---|---|---|---|---|
| Credit | Selected features | 24 | 12 | 1 | 1 |
| | Testing accuracy | 0.740 | 0.684 | 0.680 | 0.680 |
| | Training accuracy | 0.706 | 0.816 | 0.706 | 0.706 |
| WineQ | Selected features | 11 | 6 | 5 | 1 |
| | Testing MSE | 0.437 | 0.500 | 0.487 | 0.635 |
| | Training MSE | 0.268 | 0.290 | 0.388 | 0.606 |

**TABLE II: Model performance for explanation-based feature selection, with different selection thresholds. $\hat{w}$ is the weight of the most important feature.**

Table II shows the results. We contrast the model performance (on training and testing metrics) in the case all features are used versus using top 50% or a variable number of features, prioritizing those with the highest total weight. This methodology identifies the most salient features, i.e., features to which much of the model performance can be attributed.

## VII. Related Work

**Continuous interpretation methods** [6, 8, 30] explain pretrained piecewise linear NNs by their linear components by observing the state and parameters of the hidden layers. Our improvements include adding completeness, i.e., extracting possible activation states in the analysis region and rejecting infeasible ones; extending this approach to arbitrary piecewise linear layers; computing feature weights for individual or pairs of outputs; and refining the analysis region with output properties. A similar technique computes the reachable output region [25, 31]. This approach can scale well to large output spaces, but sacrifices some of the information from neuron activation, causing overapproximation.

**Local post-hoc explanations** provide explanations for black-box models around a single query point through different methods such as local approximation [2, 3], post-hoc feature selection around the query point [4, 32, 33] or counter-factual examples [34]. Compared to those approaches, we do not handle black-box models; however, our proposed method is exact and does not rely on sampling as the techniques above, and explains continuous regions of the inputs rather than being limited to single instances.

**Model mimicking** relies on building an inherently interpretable model that closely approximates the one to be explained [11, 12]. However, this model usually differs from the original in some ways, which might cause lower performance. Our approach directly explains the original model.

## VIII. Conclusion

We tackled the challenging problem of explaining NNs over continuous inputs. We showed that encoding a piecewise linear NN with formal constraints enables explanations over continuous subregions of the input space, and using the parameters of a trained model allows us to extract interpretable weights for all input features and guarantees that explanations are exact.

## References


[1] Z. C. Lipton, "The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery," *Queue*, vol. 16, no. 3, 2018.

[2] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should i trust you": Explaining the Predictions of Any Classifier," in *SIGKDD*, 2016.

[3] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *NeurIPS*, 2017.

[4] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision Model-agnostic Explanations," in *AAAI*, 2018.

[5] M. Alfarra, A. Bibi, H. Hammoud, M. Gaafar, and B. Ghanem, "On the decision boundaries of deep neural networks: A tropical geometry perspective," *arXiv preprint arXiv:2002.08838*, 2020.

[6] L. Chu, X. Hu, J. Hu, L. Wang, and J. Pei, "Exact and Consistent Interpretation for Piecewise Linear Neural Networks: A Closed Form Solution," in *SIGKDD*, 2018.

[7] M. Jordan and A. G. Dimakis, "Exactly Computing the Local Lipschitz Constant of ReLU Networks," *arXiv preprint arXiv:2003.01219*, 2020.

[8] X. Zhang, A. Solar-Lezama, and R. Singh, "Interpreting Neural Network Judgments via Minimal, Stable, and Symbolic Corrections," *NeurIPS*, 2018.

[9] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *CAV*, 2017.

[10] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The Marabou Framework for Verification and Analysis of Deep Neural Networks," in *CAV*, 2019.

[11] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting Deep Learning-Based Networking Systems," in *SIGCOMM*, 2020.

[12] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically Interpretable Reinforcement Learning," in *ICML*, 2018.

[13] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju, "Fooling LIME and SHAP: Adversarial Attacks on Post Hoc Explanation Methods," in *AIES*, 2020.

[14] L. Pulina and A. Tacchella, "Challenging SMT solvers to verify neural networks," *AI Communications*, vol. 25, 2012.

[15] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating Robustness of Neural Networks with Mixed Integer Programming," in *ICLR*, 2019.

[16] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. P. Vielma, "Strong mixed-integer programming formulations for trained neural networks," *Mathematical Programming*, vol. 183, 2020.

[17] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *ICLR*, 2019.

[18] M. Fischetti and J. Jo, "Deep Neural Networks and Mixed Integer Linear Optimization," *Constraints*, vol. 23, no. 3, 2018.

[19] H. Mao, R. Netravali, and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," in *SIGCOMM*, 2017.

[20] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling Deep Reinforcement Learning for Datacenter-scale Automatic Traffic Optimization," in *SIGCOMM*, 2018.

[21] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A Deep Reinforcement Learning perspective on Internet Congestion Control," in *ICML*, 2019.

[22] S. Fu, S. Gupta, R. Mittal, and S. Ratnasamy, "On the Use of ML for Blackbox System Performance Prediction," in *NSDI*, 2021.

[23] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI$^2$: Safety and Robustness Certification of Neural Networks with Abstract Interpretation," in *IEEE S&P*, 2018.

[24] M. Fischetti, A. Lodi, and G. Zarpellon, "Learning MILP resolution outcomes before reaching time-limit," in *CPAIOR*, 2019.

[25] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output Reachable Set Estimation and Verification for Multilayer Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, 2018.

[26] A. Dethise, M. Canini, and N. Narodytska, "Analyzing Learning-Based Networked Systems with Formal Verification," in *INFOCOM*, 2021.

[27] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade*. Springer, 2012.

[28] H. Hofmann, "Statlog (german credit data) data set," *UCI Repository of Machine Learning Databases*, vol. 53, 1994.

[29] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, 2009.

[30] D. Gopinath, H. Converse, C. Pasareanu, and A. Taly, "Property Inference for Deep Neural Networks," in *IEEE/ACM Conference on Automated Software Engineering (ASE)*, 2019.

[31] M. Mirman, T. Gehr, and M. Vechev, "Differentiable Abstract Interpretation for Provably Robust Neural Networks," in *ICML*, 2018.

[32] J. Chen, L. Song, M. Wainwright, and M. Jordan, "Learning to Explain: An Information-Theoretic Perspective on Model Interpretation," in *ICML*, 2018.

[33] P. W. Koh and P. Liang, "Understanding Black-Box Predictions via Influence Functions," in *ICML*, 2017.

[34] R. K. Mothilal, A. Sharma, and C. Tan, "Explaining Machine Learning Classifiers through Diverse Counterfactual Explanations," in *FAT**, 2020.