Exact and Consistent Interpretation for Piecewise Linear Neural Networks: A Closed Form Solution*

Lingyang Chu
Simon Fraser University
Burnaby, Canada
lca117@sfu.ca

Xia Hu
Simon Fraser University
Burnaby, Canada
huxiah@sfu.ca

Juhua Hu
Simon Fraser University
Burnaby, Canada
juhuah@sfu.ca

Lanjun Wang
Huawei Technology Co. Ltd
Beijing, China
lanjun.wang@huawei.com

Jian Pei
JD.com and Simon Fraser University
Beijing/Burnaby, China/Canada
jpei@cs.sfu.ca

ABSTRACT

Strong intelligent machines powered by deep neural networks are increasingly deployed as black boxes to make decisions in risk-sensitive domains, such as finance and medical. To reduce potential risk and build trust with users, it is critical to interpret how such machines make their decisions. Existing works interpret a pre-trained neural network by analyzing hidden neurons, mimicking pre-trained models or approximating local predictions. However, these methods do not provide a guarantee on the exactness and consistency of their interpretations. In this paper, we propose an elegant closed form solution named *OpenBox* to compute exact and consistent interpretations for the family of Piecewise Linear Neural Networks (PLNN). The major idea is to first transform a PLNN into a mathematically equivalent set of linear classifiers, then interpret each linear classifier by the features that dominate its prediction. We further apply *OpenBox* to demonstrate the effectiveness of non-negative and sparse constraints on improving the interpretability of PLNNs. The extensive experiments on both synthetic and real world data sets clearly demonstrate the exactness and consistency of our interpretation.

KEYWORDS

Deep neural network; exact and consistent interpretation; closed form.

ACM Reference Format:
Lingyang Chu, Xia Hu, Juhua Hu, Lanjun Wang, and Jian Pei . 2018. Exact and Consistent Interpretation for Piecewise Linear Neural Networks: A Closed Form Solution. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220063

*This work was supported in part by the NSERC Discovery Grant program, the Canada Research Chair program, the NSERC Strategic Grant program. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220063

1 INTRODUCTION

More and more machine learning systems are making significant decisions routinely in important domains, such as medical practice, autonomous driving, criminal justice, and military decision making [15]. As the impact of machine-made decisions increases, the demand on clear interpretations of machine learning systems is growing ever stronger against the blind deployments of decision machines [17]. Accurately and reliably interpreting a machine learning model is the key to many significant tasks, such as identifying failure models [1], building trust with human users [35], discovering new knowledge [34], and avoiding unfairness issues [45].

The interpretation problem of machine learning models has been studied for decades. Conventional models, such as Logistic Regression and Support Vector Machine, have all been well interpreted from both practical and theoretical perspectives [4]. Powerful non-negative and sparse constraints are also developed to enhance the interpretability of conventional models by sparse feature selection [21, 27]. However, due to the complex network structure of a deep neural network, the interpretation problem of modern deep models is yet a challenging field that awaits further exploration.

As to be reviewed in Section 2, the existing studies interpret a deep neural network in three major ways. The hidden neuron analysis methods [9, 29, 44] analyze and visualize the features learned by the hidden neurons of a neural network; the model mimicking methods [2, 3, 7, 20] build a transparent model to imitate the classification function of a deep neural network; the local explanation methods [11, 37, 39, 41] study the predictions on local perturbations of an input instance, so as to provide decision features for interpretation. All these methods gain useful insights into the mechanism of deep models. However, there is no guarantee that what they compute as an interpretation is truthfully the exact behavior of a deep neural network. As demonstrated by Ghorbani [13], most existing interpretation methods are inconsistent and fragile, because two perceptively indistinguishable instances with the same prediction result can be easily manipulated to have dramatically different interpretations.

Can we compute an exact and consistent interpretation for a pre-trained deep neural network? In this paper, we provide an affirmative answer, as well as an elegant closed form solution for the

family of piecewise linear neural networks. Here, a **piecewise linear neural network** (**PLNN**) [18] is a neural network that adopts a piecewise linear activation function, such as MaxOut [16] and the family of ReLU [14, 19, 31]. The wide applications [26] and great practical successes [25] of PLNNs call for exact and consistent interpretations on the overall behaviour of this type of neural networks. We make the following technical contributions.

First, we prove that a PLNN is mathematically equivalent to a set of local linear classifiers, each of which being a linear classifier that classifies a group of instances within a convex polytope in the input space. Second, we propose a method named *OpenBox* to provide an exact interpretation of a PLNN by computing its equivalent set of local linear classifiers in closed form. Third, we interpret the classification result of each instance by the decision features of its local linear classifier. Since all instances in the same convex polytope share the same local linear classifier, our interpretations are consistent per convex polytope. Fourth, we also apply *OpenBox* to study the effect of non-negative and sparse constraints on the interpretability of PLNNs. We find that a PLNN trained with these constraints selects meaningful features that dramatically improve the interpretability. Last, we conduct extensive experiments on both synthetic and real-world data sets to verify the effectiveness of our method.

The rest of this paper is organized as follows. We review the related works in Section 2. We formulate the problem in Section 3 and present *OpenBox* in Section 4. We report the experimental results in Section 5, and conclude the paper in Section 6.

2 RELATED WORKS

How to interpret the overall mechanism of deep neural networks is an emergent and challenging problem.

2.1 Hidden Neuron Analysis Methods

The hidden neuron analysis methods [9, 29, 44] interpret a pre-trained deep neural network by visualizing, revert-mapping or labeling the features that are learned by the hidden neurons.

Yosinski *et al.* [44] visualized the live activations of the hidden neurons of a ConvNet, and proposed a regularized optimization to produce a qualitatively better visualization. Erhan *et al.* [10] proposed an activation maximization method and a unit sampling method to visualize the features learned by hidden neurons. Cao *et al.* [5] visualized a neural network's attention on its target objects by a feedback loop that infers the activation status of the hidden neurons. Li *et al.* [28] visualized the compositionality of clauses by analyzing the outputs of hidden neurons in a neural model for Natural Language Processing.

To understand the features learned by the hidden neurons, Mahendran *et al.* [29] proposed a general framework that revert-maps the features learned from an image to reconstruct the image. Dosovitskiy *et al.* [9] performed the same task as Mahendran *et al.* [29] did by training an up-convolutional neural network.

Zhou *et al.* [46] interpreted a CNN by labeling each hidden neuron with a best aligned human-understandable semantic concept. However, it is hard to get a golden dataset with accurate and complete labels of all human semantic concepts.

The hidden neuron analysis methods provide useful qualitative insights into the properties of each hidden neuron. However, qualitatively analyzing every neuron does not provide much actionable and quantitative interpretation about the overall mechanism of the entire neural network [12].

2.2 Model Mimicking Methods

By imitating the classification function of a neural network, the model mimicking methods [2, 3, 7, 20] build a transparent model that is easy to interpret and achieves a high classification accuracy.

Ba *et al.* [2] proposed a model compression method to train a shallow mimic network using the training instances labeled by one or more deep neural networks. Hinton *et al.* [20] proposed a distillation method that distills the knowledge of a large neural network by training a relatively smaller network to mimic the prediction probabilities of the original large network. To improve the interpretability of distilled knowledge, Frosst and Hinton [12] extended the distillation method [20] by training a soft decision tree to mimic the prediction probabilities of a deep neural network.

Che *et al.* [7] proposed a mimic learning method to learn interpretable phenotype features. Wu *et al.* [42] proposed a tree regularization method that uses a binary decision tree to mimic and regularize the classification function of a deep time-series model. Zhu *et al.* [48] built a transparent forest model on top of a deep feature embedding network, however it is still difficult to interpret the deep feature embedding network.

The mimic models built by model mimicking methods are much simpler to interpret than deep neural networks. However, due to the reduced model complexity of a mimic model, there is no guarantee that a deep neural network with a large VC-dimension [18, 24, 40] can be successfully imitated by a simpler shallow model. Thus, there is always a gap between the interpretation of a mimic model and the actual overall mechanism of the target deep neural network.

2.3 Local Interpretation Methods

The local interpretation methods [11, 37, 39, 41] compute and visualize the important features for an input instance by analyzing the predictions of its local perturbations.

Simonyan *et al.* [38] generated a class-representative image and a class-saliency map for each class of images by computing the gradient of the class score with respect to an input image. Ribeiro *et al.* [35] proposed LIME to interpret the predictions of any classifier by learning an interpretable model in the local region around the input instance.

Zhou *et al.* [47] proposed CAM to identify discriminative image regions for each class of images using the global average pooling in CNNs. Selvaraju *et al.* [36] generalized CAM [47] by Grad-CAM, which identifies important regions of an image by flowing class-specific gradients into the final convolutional layer of a CNN.

Koh *et al.* [23] used influence functions to trace a model's prediction and identify the training instances that are the most responsible for the prediction.

The local interpretation methods generate an insightful individual interpretation for each input instance. However, the interpretations for perspectively indistinguishable instances may not be

consistent [13], and can be manipulated by a simple transformation of the input instance without affecting the prediction result [22].

3 PROBLEM DEFINITION

For a PLNN $\mathcal{N}$ that contains L layers of neurons, we write the l-th layer of $\mathcal{N}$ as $\mathcal{L}_l$. Hence, $\mathcal{L}_1$ is the **input layer**, $\mathcal{L}_L$ is the **output layer**, and the other layers $\mathcal{L}_l, l \in \{2, \ldots, L-1\}$ are **hidden layers**. A neuron in a hidden layer is called a **hidden neuron**. Let n_l represent the number of neurons in $\mathcal{L}_l$, the total number of hidden neurons in $\mathcal{N}$ is computed by $N = \sum_{l=2}^{L-1} n_l$.

Denote by $\mathbf{u}_i^{(l)}$ the i-th neuron in $\mathcal{L}_l$, by $\mathbf{b}_i^{(l-1)}$ its bias, by $\mathbf{a}_i^{(l)}$ its output, and by $\mathbf{z}_i^{(l)}$ the total weighted sum of its inputs. For all the n_l neurons in $\mathcal{L}_l$, we write their biases as a vector $\mathbf{b}^{(l-1)} = [\mathbf{b}_1^{(l-1)}, \ldots, \mathbf{b}_{n_l}^{(l-1)}]^\top$, their outputs as a vector $\mathbf{a}^{(l)} = [\mathbf{a}_1^{(l)}, \ldots, \mathbf{a}_{n_l}^{(l)}]^\top$, and their inputs as a vector $\mathbf{z}^{(l)} = [\mathbf{z}_1^{(l)}, \ldots, \mathbf{z}_{n_l}^{(l)}]^\top$.

Neurons in successive layers are connected by weighted edges. Denote by $W_{ij}^{(l)}$ the weight of the edge between the i-th neuron in $\mathcal{L}_{l+1}$ and the j-th neuron in $\mathcal{L}_l$, that is, $W^{(l)}$ is an n_{l+1}-by-n_l matrix. For $l \in \{1, \ldots, L-1\}$, we compute $\mathbf{z}^{(l+1)}$ by

$$\mathbf{z}^{(l+1)} = W^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l)} \tag{1}$$

Denote by $f : \mathbb{R} \to \mathbb{R}$ the piecewise linear activation function for each neuron in the hidden layers of $\mathcal{N}$. We have $\mathbf{a}_i^{(l)} = f(\mathbf{z}_i^{(l)})$ for all $l \in \{2, \ldots, L-1\}$. We extend f to apply to vectors in an element-wise fashion, such that $f(\mathbf{z}^{(l)}) = [f(\mathbf{z}_1^{(l)}), \ldots, f(\mathbf{z}_{n_l}^{(l)})]^\top$. Then, we compute $\mathbf{a}^{(l)}$ for all $l \in \{2, \ldots, L-1\}$ by

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) \tag{2}$$

An **input instance** of $\mathcal{N}$ is denoted by $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X} \subseteq \mathbb{R}^d$ is a d-dimensional input space. $\mathbf{x}$ is also called an **instance** for short.

Denote by $\mathbf{x}_i$ the i-th dimension of $\mathbf{x}$. The input layer $\mathcal{L}_1$ contains $n_1 = d$ neurons, where $\mathbf{a}_i^{(1)} = \mathbf{x}_i$ for all $i \in \{1, \ldots, d\}$.

The **output** of $\mathcal{N}$ is $\mathbf{a}^{(L)} \in \mathcal{Y}$, where $\mathcal{Y} \subseteq \mathbb{R}^{n_L}$ is an n_L-dimensional output space. The output layer $\mathcal{L}_L$ adopts the *softmax* function to compute the output by $\mathbf{a}^{(L)} = softmax(\mathbf{z}^{(L)})$.

A PLNN works as a **classification function** $F : \mathcal{X} \to \mathcal{Y}$ that maps an input $\mathbf{x} \in \mathcal{X}$ to an output $\mathbf{a}^{(L)} \in \mathcal{Y}$. It is widely known that $F(\cdot)$ is a piecewise linear function [30, 33]. However, due to the complex network of a PLNN, the overall behaviour of $F(\cdot)$ is hard to understand. Thus, a PLNN is usually regarded as a black box.

How to interpret the overall behavior of a PLNN in a human-understandable manner is an interesting problem that has attracted much attention in recent years.

Following a principled approach of interpreting a machine learning model [4], we regard an **interpretation** of a PLNN $\mathcal{N}$ as the decision features that define the decision boundary of $\mathcal{N}$. We call a model **interpretable** if it explicitly provides its interpretation (i.e., decision features) in closed form.

Definition 3.1. Given a fixed PLNN $\mathcal{N}$ with constant structure and parameters, our task is to interpret the overall behaviour of $\mathcal{N}$ by computing an interpretable model $\mathcal{M}$ that satisfies the following requirements.

Table 1: Frequently used notations.

Notation	Description
$\mathbf{u}_i^{(l)}$	The i-th neuron in layer $\mathcal{L}_l$.
n_l	The number of neurons in layer $\mathcal{L}_l$.
N	The total number of hidden neurons in $\mathcal{N}$.
$\mathbf{z}_i^{(l)}$	The input of the i-th neuron in layer $\mathcal{L}_l$.
$\mathbf{c}_i^{(l)}$	The configuration of the i-th neuron in layer $\mathcal{L}_l$.
C_h	The h-th configuration of the PLNN $\mathcal{N}$.
P_h	The h-th convex polytope determined by C_h.
$F_h(\cdot)$	The h-th linear classifier that is determined by C_h.
Q_h	The set of linear inequalities that define P_h.

- **Exactness:** $\mathcal{M}$ is mathematically equivalent to $\mathcal{N}$ such that the interpretations provided by $\mathcal{M}$ truthfully describe the exact behaviour of $\mathcal{N}$.
- **Consistency:** $\mathcal{M}$ provides similar interpretations for classification of similar instances.

Table 1 summarizes a list of frequently used notations.

4 THE OPENBOX METHOD

In this section, we describe the *OpenBox* method, which produces an exact and consistent interpretation of a PLNN by computing an interpretation model $\mathcal{M}$ in a piecewise linear closed form.

We first define the configuration of a PLNN $\mathcal{N}$, which specifies the activation status of each hidden neuron in $\mathcal{N}$. Then, we illustrate how to interpret the classification result of a fixed instance. Last, we illustrate how to interpret the overall behavior of $\mathcal{N}$ by computing an interpretation model $\mathcal{M}$ that is mathematically equivalent to $\mathcal{N}$.

4.1 The Configuration of a PLNN

For a hidden neuron $\mathbf{u}_i^{(l)}$, the piecewise linear activation function $f(\mathbf{z}_i^{(l)})$ is in the following form.

$$f(\mathbf{z}_i^{(l)}) = \begin{cases} r_1\mathbf{z}_i^{(l)} + t_1, & \text{if } \mathbf{z}_i^{(l)} \in I_1 \\ r_2\mathbf{z}_i^{(l)} + t_2, & \text{if } \mathbf{z}_i^{(l)} \in I_2 \\ \quad\vdots & \\ r_k\mathbf{z}_i^{(l)} + t_k, & \text{if } \mathbf{z}_i^{(l)} \in I_k \end{cases} \tag{3}$$

where $k \geq 1$ is a constant integer, $f(\mathbf{z}_i^{(l)})$ consists of k linear functions, $\{r_1, \ldots, r_k\}$ are constant **slopes**, $\{t_1, \ldots, t_k\}$ are constant **intercepts**, and $\{I_1, \ldots, I_k\}$ is a collection of constant **real intervals** that partition $\mathbb{R}$.

Given a fixed PLNN $\mathcal{N}$, an instance $\mathbf{x} \in \mathcal{X}$ determines the value of $\mathbf{z}_i^{(l)}$, and further determines a linear function in $f(\mathbf{z}_i^{(l)})$ to apply. According to which linear function in $f(\mathbf{z}_i^{(l)})$ is applied, we encode the activation status of each hidden neuron by k **states**, each of which uniquely corresponds to one of the k linear functions of $f(\mathbf{z}_i^{(l)})$. Denote by $\mathbf{c}_i^{(l)} \in \{1, \ldots, k\}$ the state of $\mathbf{u}_i^{(l)}$, we have $\mathbf{z}_i^{(l)} \in I_q$ if and only if $\mathbf{c}_i^{(l)} = q$ ($q \in \{1, \ldots, k\}$). Since the inputs $\mathbf{z}_i^{(l)}$'s are different from neuron to neuron, the states of different hidden neurons may differ from each other.

Denote by a vector $\mathbf{c}^{(l)} = [\mathbf{c}_1^{(l)}, \ldots, \mathbf{c}_{n_l}^{(l)}]$ the states of all hidden neurons in $\mathcal{L}_l$. The **configuration** of $\mathcal{N}$ is an N-dimensional vector, denoted by $\mathbf{C} = [\mathbf{c}^{(2)}, \ldots, \mathbf{c}^{(L-1)}]$, which specifies the states of all hidden neurons in $\mathcal{N}$.

The configuration $\mathbf{C}$ of a fixed PLNN is uniquely determined by the instance $\mathbf{x}$. We write the function that maps an instance $\mathbf{x} \in \mathcal{X}$ to a configuration $\mathbf{C} \in \{1, \ldots, k\}^N$ as $conf : \mathcal{X} \to \{1, \ldots, k\}^N$.

For a neuron $\mathbf{u}_i^{(l)}$, denote by variables $\mathbf{r}_i^{(l)}$ and $\mathbf{t}_i^{(l)}$ the slope and intercept, respectively, of the linear function that corresponds to the state $\mathbf{c}_i^{(l)}$. $\mathbf{r}_i^{(l)}$ and $\mathbf{t}_i^{(l)}$ are uniquely determined by $\mathbf{c}_i^{(l)}$, such that $\mathbf{r}_i^{(l)} = r_q$ and $\mathbf{t}_i^{(l)} = t_q$, if and only if $\mathbf{c}_i^{(l)} = q$ ($q \in \{1, \ldots, k\}$).

For all hidden neurons in $\mathcal{L}_l$, we write the variables of slopes and intercepts as $\mathbf{r}^{(l)} = [\mathbf{r}_1^{(l)}, \ldots, \mathbf{r}_{n_l}^{(l)}]^\top$ and $\mathbf{t}^{(l)} = [\mathbf{t}_1^{(l)}, \ldots, \mathbf{t}_{n_l}^{(l)}]^\top$, respectively. Then, we rewrite the activation function for all neurons in a hidden layer $\mathcal{L}_l$ as

$$f(\mathbf{z}^{(l)}) = \mathbf{r}^{(l)} \circ \mathbf{z}^{(l)} + \mathbf{t}^{(l)} \tag{4}$$

where $\mathbf{r}^{(l)} \circ \mathbf{z}^{(l)}$ is the Hadamard product between $\mathbf{r}^{(l)}$ and $\mathbf{z}^{(l)}$.

Next, we interpret the classification result of a fixed instance.

4.2 Exact Interpretation for the Classification Result of a Fixed Instance

Given a fixed PLNN $\mathcal{N}$, we interpret the classification result of a fixed instance $\mathbf{x} \in \mathcal{X}$ by deriving the closed form of $F(\mathbf{x})$ as follows.

Following Equations 2 and 4, we have, for all $l \in \{2, \ldots, L-1\}$

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) = \mathbf{r}^{(l)} \circ \mathbf{z}^{(l)} + \mathbf{t}^{(l)}$$

By plugging $\mathbf{a}^{(l)}$ into Equation 1, we rewrite $\mathbf{z}^{(l+1)}$ as

$$\mathbf{z}^{(l+1)} = W^{(l)}(\mathbf{r}^{(l)} \circ \mathbf{z}^{(l)} + \mathbf{t}^{(l)}) + \mathbf{b}^{(l)} = \tilde{W}^{(l)}\mathbf{z}^{(l)} + \tilde{\mathbf{b}}^{(l)} \tag{5}$$

where $\tilde{\mathbf{b}}^{(l)} = W^{(l)}\mathbf{t}^{(l)} + \mathbf{b}^{(l)}$, and $\tilde{W}^{(l)} = W^{(l)} \circ \mathbf{r}^{(l)}$ is an extended version of Hadamard product, such that the entry at the i-th row and j-th column of $\tilde{W}^{(l)}$ is $\tilde{W}_{ij}^{(l)} = W_{ij}^{(l)}\mathbf{r}_j^{(l)}$.

By iteratively plugging Equation 5 into itself, we can write $\mathbf{z}^{(l+1)}$ for all $l \in \{2, \ldots, L-1\}$ as

$$\mathbf{z}^{(l+1)} = \prod_{h=0}^{l-2} \tilde{W}^{(l-h)}\mathbf{z}^{(2)} + \sum_{h=2}^{l} \prod_{q=0}^{l-h-1} \tilde{W}^{(l-q)}\tilde{\mathbf{b}}^{(h)}$$

By plugging $\mathbf{z}^{(2)} = W^{(1)}\mathbf{a}^{(1)} + \mathbf{b}^{(1)}$ and $\mathbf{a}^{(1)} = \mathbf{x}$ into the above equation, we rewrite $\mathbf{z}^{(l+1)}$, for all $l \in \{2, \ldots, L-1\}$, as

$$\begin{aligned}\mathbf{z}^{(l+1)} &= \prod_{h=0}^{l-2} \tilde{W}^{(l-h)}W^{(1)}\mathbf{x} + \prod_{h=0}^{l-2} \tilde{W}^{(l-h)}\mathbf{b}^{(1)} + \sum_{h=2}^{l} \prod_{q=0}^{l-h-1} \tilde{W}^{(l-q)}\tilde{\mathbf{b}}^{(h)} \\ &= \hat{W}^{(1:l)}\mathbf{x} + \hat{\mathbf{b}}^{(1:l)}\end{aligned} \tag{6}$$

where $\hat{W}^{(1:l)} = \prod_{h=0}^{l-2} \tilde{W}^{(l-h)}W^{(1)}$ is the coefficient matrix of $\mathbf{x}$, and $\hat{\mathbf{b}}^{(1:l)}$ is the sum of the remaining terms. The superscript $(1:l)$ indicates that $\hat{W}^{(1:l)}\mathbf{x} + \hat{\mathbf{b}}^{(1:l)}$ is equivalent to PLNN's forward propagation from layer $\mathcal{L}_1$ to layer $\mathcal{L}_l$.

Since the output of $\mathcal{N}$ on an input $\mathbf{x} \in \mathcal{X}$ is $F(\mathbf{x}) = \mathbf{a}^{(L)} = softmax(\mathbf{z}^{(L)})$, the **closed form** of $F(\mathbf{x})$ is

$$F(\mathbf{x}) = softmax(\hat{W}^{(1:L-1)}\mathbf{x} + \hat{\mathbf{b}}^{(1:L-1)}) \tag{7}$$

For a fixed PLNN $\mathcal{N}$ and a fixed instance $\mathbf{x}$, $\hat{W}^{(1:L-1)}$ and $\hat{\mathbf{b}}^{(1:L-1)}$ are constant parameters uniquely determined by the fixed configuration $\mathbf{C} = conf(\mathbf{x})$. Therefore, for a fixed input instance $\mathbf{x}$, $F(\mathbf{x})$ is a **linear classifier** whose decision boundary is explicitly defined by $\hat{W}^{(1:L-1)}\mathbf{x} + \hat{\mathbf{b}}^{(1:L-1)}$.

Inspired by the interpretation method widely used by conventional linear classifiers, such as Logistic Regression and linear SVM [4], we interpret the prediction on a fixed instance $\mathbf{x}$ by the decision features of $F(\mathbf{x})$. Specifically, the entries of the i-th row of $\hat{W}^{(1:L-1)}$ are the **decision features** for the i-th class of instances.

Equation 7 provides a straightforward way to interpret the classification result of a fixed instance. However, individually interpreting the classification result of every single instance is far from the understanding of the overall behavior of a PLNN $\mathcal{N}$. Next, we describe how to interpret the overall behavior of $\mathcal{N}$ by computing an interpretation model $\mathcal{M}$ that is mathematically equivalent to $\mathcal{N}$.

4.3 Exact Interpretation of a PLNN

A fixed PLNN $\mathcal{N}$ with N hidden neurons has at most k^N configurations. We represent the h-th configuration by $\mathbf{C}_h \in C$, where $C \subseteq \{1, \ldots, k\}^N$ is the set of all configurations of $\mathcal{N}$.

Recall that each instance $\mathbf{x} \in \mathcal{X}$ uniquely determines a configuration $conf(\mathbf{x}) \in C$. Since the volume of C, denoted by $|C|$, is at most k^N, but the number of instances in $\mathcal{X}$ can be arbitrarily large, it is clear that at least one configuration in C should be shared by more than one instances in $\mathcal{X}$.

Denote by $P_h = \{\mathbf{x} \in \mathcal{X} \mid conf(\mathbf{x}) = \mathbf{C}_h\}$ the set of instances that have the same configuration $\mathbf{C}_h$. We prove in Theorem 4.1 that for any configuration $\mathbf{C}_h \in C$, P_h is a convex polytope in $\mathcal{X}$.

Theorem 4.1. *Given a fixed PLNN $\mathcal{N}$ with N hidden neurons, $\forall \mathbf{C}_h \in C$, $P_h = \{\mathbf{x} \in \mathcal{X} \mid \text{conf}(\mathbf{x}) = \mathbf{C}_h\}$ is a convex polytope in $\mathcal{X}$.*

Proof. We prove by showing that $conf(\mathbf{x}) = \mathbf{C}_h$ is equivalent to a finite set of linear inequalities with respect to $\mathbf{x}$.

When $l = 2$, we have $\mathbf{z}^{(2)} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$. For $l \in \{3, \ldots, L-1\}$, it follows Equation 6 that $\mathbf{z}^{(l)} = \hat{W}^{(1:l-1)}\mathbf{x} + \hat{\mathbf{b}}^{(1:l-1)}$, which is a linear function of $\mathbf{x}$, because $\hat{W}^{(1:l)}$ and $\hat{\mathbf{b}}^{(1:l)}$ are constant parameters when $\mathbf{C}_h$ is fixed. In summary, given a fixed $\mathbf{C}_h$, $\mathbf{z}^{(l)}$ is a linear function of $\mathbf{x}$ for all $l \in \{2, \ldots, L-1\}$.

We show that P_h is a convex polytope by showing that $conf(\mathbf{x}) = \mathbf{C}_h$ is equivalent to a set of $2N$ linear inequalities with respect to $\mathbf{x}$. Recall that $\mathbf{z}_i^{(l)} \in I_q$ if and only if $\mathbf{c}_i^{(l)} = q$ ($q \in \{1, \ldots, k\}$). Denote by $\psi : \{1, \ldots, k\} \to \{I_1, \ldots, I_k\}$ the bijective function that maps a configuration $\mathbf{c}_i^{(l)}$ to a real interval in $\{I_1, \ldots, I_k\}$, such that $\psi(\mathbf{c}_i^{(l)}) = I_q$ if and only if $\mathbf{c}_i^{(l)} = q$ ($q \in \{1, \ldots, k\}$). Then, $conf(\mathbf{x}) = \mathbf{C}_h$ is equivalent to a set of constraints, denoted by $Q_h = \{\mathbf{z}_i^{(l)} \in \psi(\mathbf{c}_i^{(l)}) \mid i \in \{1, \ldots, n_l\}, l \in \{2, \ldots, L-1\}\}$. Since $\mathbf{z}_i^{(l)}$ is a linear function of $\mathbf{x}$ and $\psi(\mathbf{c}_i^{(l)})$ is a real interval, each constraint $\mathbf{z}_i^{(l)} \in \psi(\mathbf{c}_i^{(l)})$ in Q_h is equivalent to two linear inequalities with respect to $\mathbf{x}$. Therefore, $conf(\mathbf{x}) = \mathbf{C}_h$ is equivalent to a set of $2N$ linear inequalities, which means P_h is a convex polytope. □

According to Theorem 4.1, all instances sharing the same configuration $\mathbf{C}_h$ form a unique convex polytope P_h that is explicitly defined by $2N$ linear inequalities in Q_h. Since $\mathbf{C}_h$ also determines

Algorithm 1: $OpenBox(\mathcal{N}, D_{\text{train}})$

Input: $\mathcal{N}$:= a fixed PLNN, $D_{\text{train}} \subset \mathcal{X}$ the set of training instances used to train $\mathcal{N}$.
Output: $\mathcal{M}$:= a set of active LLCs

1: Initialization: $\mathcal{M} = \emptyset, C = \emptyset$.
2: **for** each $\mathbf{x} \in D_{\text{train}}$ **do**
3: Compute the configuration by $\mathbf{C}_h \leftarrow conf(\mathbf{x})$.
4: **if** $\mathbf{C}_h \notin C$ **then**
5: $C \leftarrow C \cup \mathbf{C}_h$ and $\mathcal{M} \leftarrow \mathcal{M} \cup (F_h(\mathbf{x}), P_h)$.
6: **end if**
7: **end for**
8: **return** $\mathcal{M}$.

the linear classifier for a fixed instance in Equation 7, all instances in the same convex polytope P_h share the same linear classifier determined by $\mathbf{C}_h$.

Denote by $F_h(\cdot)$ the linear classifier that is shared by all instances in P_h, we can interpret $\mathcal{N}$ as a set of **local linear classifiers** (**LLCs**), each LLC being a linear classifier $F_h(\cdot)$ that applies to all instances in a convex polytope P_h. Denote by a tuple $(F_h(\cdot), P_h)$ the h-th LLC, a fixed PLNN $\mathcal{N}$ is equivalent to a set of LLCs, denoted by $\mathcal{M} = \{(F_h(\cdot), P_h) \mid \mathbf{C}_h \in C\}$. We use $\mathcal{M}$ as our final interpretation model for $\mathcal{N}$.

For a fixed PLNN $\mathcal{N}$, if the states of the N hidden neurons are independent, the PLNN $\mathcal{N}$ has k^N configurations, which means $\mathcal{M}$ contains k^N LLCs. However, due to the hierarchical structure of a PLNN, the states of a hidden neuron in $\mathcal{L}_l$ strongly correlate with the states of the neurons in the former layers $\mathcal{L}_q (q < l)$. Therefore, the volume of C is much less than k^N, and the number of local linear classifiers in $\mathcal{M}$ is much less than k^N. We discuss this phenomenon later in Table 3 and Section 5.4.

In practice, we do not need to compute the entire set of LLCs in $\mathcal{M}$ all at once. Instead, we can first compute an active subset of $\mathcal{M}$, that is, the set of LLCs that are actually used to classify the available set of instances. Then, we can update $\mathcal{M}$ whenever a new LLC is used to classify a newly coming instance.

Algorithm 1 summarizes the *OpenBox* method, which computes $\mathcal{M}$ as the active set of LLCs that are actually used to classify the set of training instances, denoted by D_{train}.

The time cost of Algorithm 1 consists of the time T_{conf} to compute $conf(\mathbf{x})$ in step 3 and the time T_{LLC} to compute the LLC $(F_h(\mathbf{x}), P_h)$ in step 5. Since T_{conf} and T_{LLC} are dominated by matrix (vector) multiplications, we evaluate the time cost of Algorithm 1 by the number of scalar multiplications. First, since we compute $conf(\mathbf{x})$ by forward propagating from layer $\mathcal{L}_1$ to layer $\mathcal{L}_{L-1}$, $T_{conf} = \sum_{l=2}^{L-1} n_l n_{l-1}$. Second, since $(F_h(\mathbf{x}), P_h)$ is determined by the set of tuples $\mathcal{G} = \{(\hat{W}^{(1:l)}, \hat{\mathbf{b}}^{(1:l)}) \mid l \in \{1, \ldots, L-1\}\}$, T_{LLC} is the time to compute $\mathcal{G}$. Given $(\hat{W}^{(1:l-1)}, \hat{\mathbf{b}}^{(1:l-1)})$, we can compute $(\hat{W}^{(1:l)}, \hat{\mathbf{b}}^{(1:l)})$ by plugging $\mathbf{z}^{(l)} = \hat{W}^{(1:l-1)}\mathbf{x} + \hat{\mathbf{b}}^{(1:l-1)}$ (Equation 6) into Equation 5, and the time cost is $n_{l+1} n_l (n_1 + 1)$. Since $\hat{W}^{(1:1)} = W^{(1)}$ and $\hat{\mathbf{b}}^{(1:1)} = \mathbf{b}^{(1)}$, we can iteratively compute $\mathcal{G}$. The overall time cost is $T_{\text{LLC}} = \sum_{l=2}^{L-1} n_{l+1} n_l (n_1 + 1)$.

The worst case of Algorithm 1 happens when every instance $\mathbf{x} \in D_{\text{train}}$ has a unique configuration $conf(\mathbf{x})$. Denote by $|D_{\text{train}}|$ the number of training instances, the time cost of Algorithm 1 in the worst case is $|D_{\text{train}}|(T_{conf} + T_{LLC})$. Since $n_l, l \in \{2, \ldots, L-1\}$ are constants and $n_1 = d$ is the size of the input $\mathbf{x} \in \mathbb{R}^d$, the time complexity of Algorithm 1 is $O(|D_{\text{train}}|d)$.

Now, we are ready to introduce how to interpret the classification result of an instance $\mathbf{x} \in P_h, h \in \{1, \ldots, |C|\}$. First, we interpret the classification result of $\mathbf{x}$ using the decision features of $F_h(\mathbf{x})$ (Section 4.2). Second, we interpret why $\mathbf{x}$ is contained in P_h using the **polytope boundary features** (**PBFs**), which are the decision features of the polytope boundaries. More specifically, a polytope boundary of P_h is defined by a linear inequality $\mathbf{z}_i^{(l)} \in \psi(\mathbf{c}_i^{(l)})$ in Q_h. By Equation 6, $\mathbf{z}_i^{(l)}$ is a linear function with respect to $\mathbf{x}$. The PBFs are the coefficients of $\mathbf{x}$ in $\mathbf{z}_i^{(l)}$.

We also discover that some linear inequalities in Q_h are redundant whose hyperplanes do not intersect with P_h. To simplify our interpretation on the polytope boundaries, we remove such redundant inequalities by Caron's method [6] and focus on studying the PBFs of the non-redundant ones.

The advantages of *OpenBox* are three-fold as follows. First, our interpretation is exact, because the set of LLCs in $\mathcal{M}$ are mathematically equivalent to the classification function $F(\cdot)$ of $\mathcal{N}$. Second, our interpretation is group-wise consistent. It is due to the reason that all instances in the same convex polytope are classified by exactly the same LLC, and thus the interpretations are consistent with respect to a given convex polytope. Last, our interpretation is easy to compute due to the low time complexity of Algorithm 1.

5 EXPERIMENTS

In this section, we evaluate the performance of *OpenBox*, and compare it with the state-of-the-art method LIME [35]. In particular, we address the following questions: (1) What are the LLCs look like? (2) Are the interpretations produced by LIME and *OpenBox* exact and consistent? (3) Are the decision features of LLCs easy to understand, and can we improve the interpretability of these features by non-negative and sparse constraints? (4) How to interpret the PBFs of LLCs? (5) How effective are the interpretations of *OpenBox* in hacking and debugging a PLNN model?

Table 2 shows the details of the six models we used. For both PLNN and PLNN-NS, we use the same network structure described in Table 3, and adopt the widely used activation function: ReLU [14]. We apply the non-negative and sparse constraints proposed by Chorowski *et al.* [8] to train PLNN-NS. Since our goal is to comprehensively study the interpretation effectiveness of *OpenBox* rather than achieving state-of-the-art classification performance, we use relatively simple network structures for PLNN and PLNN-NS, which are still powerful enough to achieve significantly better classification performance than Logistic Regression (LR). The decision features of LR, LR-F, LR-NS and LR-NSF are used as baselines to compare with the decision features of LLCs.

The Python code of LIME is published by its authors[1]. The other methods and models are implemented in Matlab. PLNN and PLNN-NS are trained using the DeepLearnToolBox [32]. All experiments are conducted on a PC with a Core-i7-3370 CPU (3.40 GHz), 16GB main memory, and a 5,400 rpm hard drive running Windows 7 OS.

[1] https://github.com/marcotcr/lime

Table 2: The models to interpret. LR is Logistic Regression. NS means non-negative and sparse constraints. Flip means the model is trained on the instances with flipped labels.

Models	PLNN	PLNN-NS	LR	LR-F	LR-NS	LR-NSF
NS	×	✓	×	×	✓	✓
Flip	×	×	×	✓	×	✓

Table 3: The network structures $(n_1, n_2, \ldots, n_L)$ and the number of configurations $|C|$ of PLNN and PLNN-NS. The neurons in successive layers are initialized to be fully connected. $k = 2$ is the number of linear functions of ReLU, N is the number of hidden neurons.

Data Sets	# Neurons $(n_1, n_2, \ldots, n_L)$	PLNN		PLNN-NS	
		$\|C\|$	k^N	$\|C\|$	k^N
SYN	(2, 4, 16, 2, 2)	266	2^{22}	41	2^{22}
FMNIST-1	(784, 8, 2, 2)	78	2^{10}	3	2^{10}
FMNIST-2	(784, 8, 2, 2)	23	2^{10}	18	2^{10}

Table 4: Detailed description of data sets.

Data Sets	Training Data		Testing Data	
	# Positive	# Negative	# Positive	# Negative
SYN	6,961	13,039	N/A	N/A
FMNIST-1	4,000	4,000	3,000	3,000
FMNIST-2	4,000	4,000	3,000	3,000

We use the following data sets. Detailed information of the data sets is shown in Table 4.

Synthetic (SYN) Data Set. As shown in Figure 1(a), this data set contains 20,000 instances uniformly sampled from a quadrangle in 2-dimensional Euclidean space. The red and blue points are positive and negative instances, respectively. Since we only use SYN to visualize the LLCs of a PLNN and we do not perform testing on SYN, we use all instances in SYN as the training data.

FMNIST-1 and FMNIST-2 Data Sets. Each of these data sets contains two classes of images in the Fashion MNIST data set [43]. FMNIST-1 consists of the images of *Ankle Boot* and *Bag*. FMNIST-2 consists of the images of *Coat* and *Pullover*. All images in FMNIST-1 and FMNIST-2 are 28-by-28 grayscale images. We represent an image by cascading the 784 pixel values into a 784-dimensional feature vector. The Fashion MNIST data set is available online[2].

5.1 What Are the LLCs Look Like?

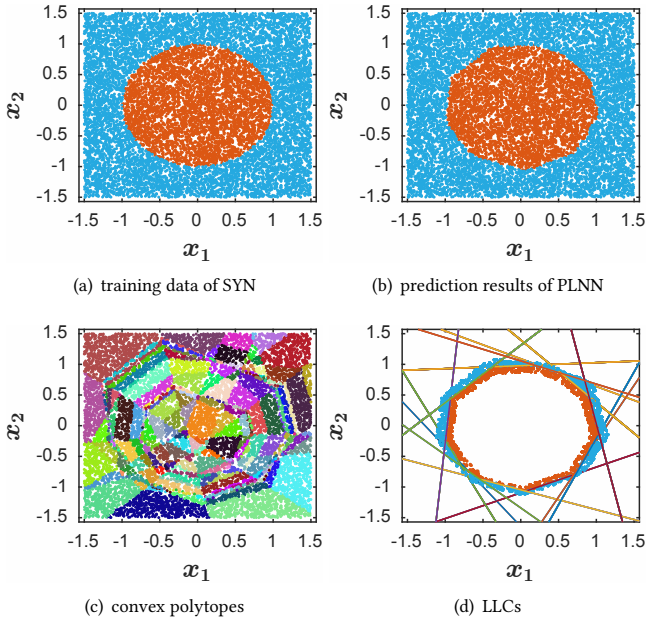
We demonstrate our claim in Theorem 4.1 by visualizing the LLCs of the PLNN trained on SYN.

Figures 1(a)-(b) show the training instances of SYN and the prediction results of PLNN on the training instances, respectively. Since all instances are used for training, the prediction accuracy is 99.9%.

In Figure 1(c), we plot all instances with the same configuration in the same colour. Clearly, all instances with the same configuration are contained in the same convex polytope. This demonstrates our claim in Theorem 4.1.

Figure 1(d) shows the LLCs whose convex polytopes cover the decision boundary of PLNN and contain both positive and negative

[2]https://github.com/zalandoresearch/fashion-mnist



(a) training data of SYN (b) prediction results of PLNN

(c) convex polytopes (d) LLCs

Figure 1: The LLCs of the PLNN trained on SYN.

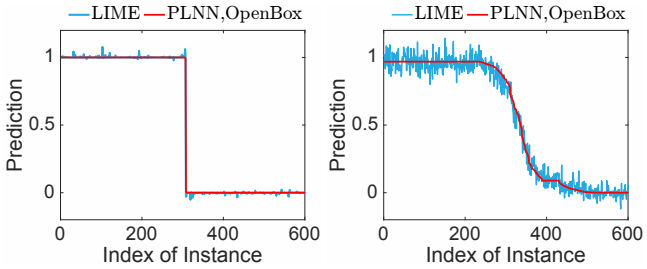
instances. As it is shown, the solid lines show the decision boundaries of the LLCs, which capture the difference between positive and negative instances, and form the overall decision boundary of PLNN. A convex polytope that does not cover the boundary of PLNN contains a single class of instances. The LLCs of these convex polytopes capture the common features of the corresponding class of instances. As to be analyzed in the following subsections, the set of LLCs produce exactly the same prediction as PLNN, and also capture meaningful decision features that are easy to understand.

5.2 Are the Interpretations Exact and Consistent?

Exact and consistent interpretations are naturally favored by human minds. In this subsection, we systematically study the exactness and consistency of the interpretations of LIME and *OpenBox* on FMNIST-1 and FMNIST-2. Since LIME is too slow to process all instances in 24 hours, for each of FMNIST-1 and FMNIST-2, we uniformly sample 600 instances from the testing set, and conduct the following experiments on the sampled instances.

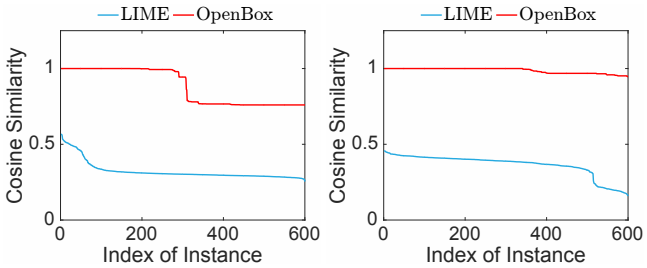
We first analyze the **exactness of interpretation** by comparing the predictions computed by the local interpretable model of LIME, the LLCs of *OpenBox* and PLNN, respectively. The prediction of an instance is the probability of classifying it as a positive instance.

In Figure 2, since LIME does not guarantee zero approximation error on the local predictions of PLNN, the predictions of LIME are not exactly the same as PLNN on FMNIST-1, and are dramatically different from PLNN on FMNIST-2. The difference of predictions is more significant on FMNIST-2, because the images in FMNIST-2 are more difficult to distinguish, which makes the decision boundary of PLNN more complicated and harder to approximate. We can also see that the predictions of LIME exceed $[0, 1]$. This is because the



(a) FMNIST-1 (b) FMNIST-2

Figure 2: The predictions of LIME, OpenBox and PLNN. The predictions of all methods are computed individually and independently. We sort the results by PLNN's predictions in descending order.



(a) FMNIST-1 (b) FMNIST-2

Figure 3: The cosine similarity between the decision features of each instance and its nearest neighbour. The results of LIME and *OpenBox* are separately sorted by cosine similarity in descending order.

output of the interpretable model of LIME is not a probability at all. As a result, it is arguable that the interpretations computed by LIME may not truthfully describe the exact behavior of PLNN. In contrast, since the set of LLCs computed by *OpenBox* is mathematically equivalent to $F(\cdot)$ of PLNN, the predictions of *OpenBox* are exactly the same as PLNN on all instances. Therefore, the decision features of LLCs exactly describe the overall behavior of PLNN.

Next, we study the **interpretation consistency** of LIME and *OpenBox* by analyzing the similarity between the interpretations of similar instances.

In general, a consistent interpretation method should provide similar interpretations for similar instances. For an instance $\mathbf{x}$, denote by $\mathbf{x}'$ the nearest neighbor of $\mathbf{x}$ by Euclidean distance, by $\gamma, \gamma' \in \mathbb{R}^d$ the decision features for the classification of $\mathbf{x}$ and $\mathbf{x}'$, respectively. We measure the consistency of interpretation by the cosine similarity between γ and γ', where a larger cosine similarity indicates a better interpretation consistency.

As shown in Figure 3, the cosine similarity of *OpenBox* is equal to 1 on about 50% of the instances, because *OpenBox* consistently gives the same interpretation for all instances in the same convex polytope. Since the nearest neighbours $\mathbf{x}$ and $\mathbf{x}'$ may not belong to the same convex polytope, the cosine similarity of *OpenBox* is not always equal to 1 on all instances. In constrast, since LIME computes individual interpretation based on the unique local perturbations of every single instance, the cosine similarity of LIME is significantly lower than *OpenBox* on all instances. This demonstrates the superior interpretation consistency of *OpenBox*.

In summary, the interpretations of *OpenBox* are exact, and are much more consistent than the interpretations of LIME.

Table 5: The training and testing accuracy of all models.

Data Set	FMNIST-1		FMNIST-2	
Accuracy	Train	Test	Train	Test
LR	0.998	0.997	0.847	0.839
LR-F	0.998	0.997	0.847	0.839
PLNN	**1.000**	**0.999**	**0.907**	**0.868**
LR-NS	0.772	0.776	0.711	0.698
LR-NSF	0.989	0.989	0.782	0.791
PLNN-NS	**1.000**	**0.999**	**0.894**	**0.867**

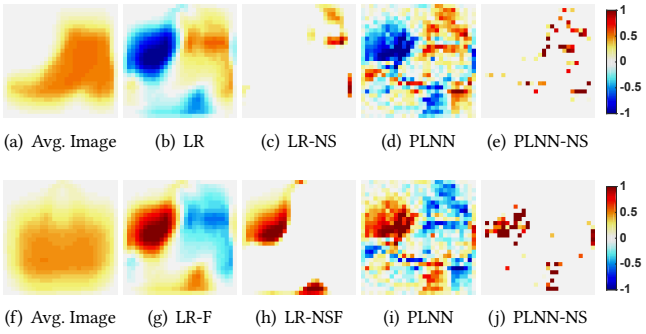
5.3 Decision Features of LLCs and the Effect of Non-negative and Sparse Constraints

Besides exactness and consistency, a good interpretation should also have a strong semantical meaning, such that the "thoughts" of an intelligent machine can be easily understood by a human brain. In this subsection, we first show the meaning of the decision features of LLCs, then study the effect of the non-negative and sparse constraints in improving the interpretability of the decision features. The decision features of PLNN and PLNN-NS are computed by *OpenBox*. The decision features of LR, LR-F, LR-NS and LR-NSF are used as baselines. Table 5 shows the accuracy of all models.

Figure 4 shows the decision features of all models on FMNIST-1. Interestingly, the decision features of PLNN are as easy to understand as the decision features of LR and LR-F. All these features clearly highlight meaningful image parts, such as the ankle and heel of *Ankle Boot*, and the upper left corner of *Bag*. A closer look at the the average images suggests that these decision features describe the difference between *Ankle Boot* and *Bag*.

The decision features of PLNN capture more detailed difference between *Ankle Boot* and *Bag* than the decision features of LR and LR-F. This is because the LLCs of PLNN only capture the difference between a subset of instances within a convex polytope, however, LR and LR-F capture the overall difference between all instances of *Ankle Boot* and *Bag*. The accuracies of PLNN, LR and LR-F are comparable because the instances of *Ankle Boot* and *Bag* are easy to distinguish. However, as to be shown in Figure 5, when the instances are hard to distinguish, PLNN captures much more detailed features than LR and LR-F, and achieves a significantly better accuracy.

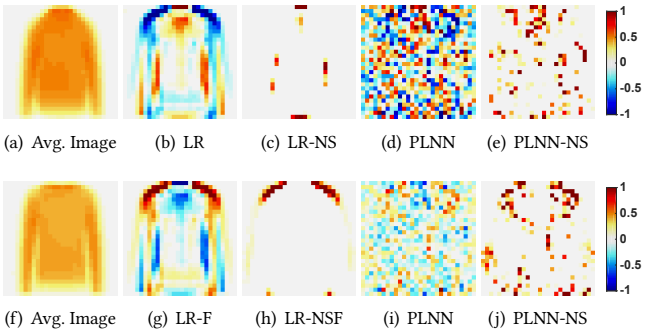
Figure 5 shows the decision features of all models on FMNIST-2. As it is shown, LR and LR-F capture decision features with a strong semantical meaning, such as the collar and breast of *Coat*, and the shoulder of *Pullover*. However, these features are too general to accurately distinguish between *Coat* and *Pullover*. Therefore, LR and LR-F do not achieve a high accuracy. Interestingly, the decision features of PLNN capture much more details than LR and LR-F, which leads to the superior accuracy of PLNN.



(a) Avg. Image (b) LR (c) LR-NS (d) PLNN (e) PLNN-NS

(f) Avg. Image (g) LR-F (h) LR-NSF (i) PLNN (j) PLNN-NS

Figure 4: The decision features of all models on FMNIST-1. (a)-(e) and (f)-(j) show the average image and the decision features of all models for *Ankle Boot* and *Bag*, respectively. For PLNN and PLNN-NS, we show the decision features of the LLC whose convex polytope contains the most instances.



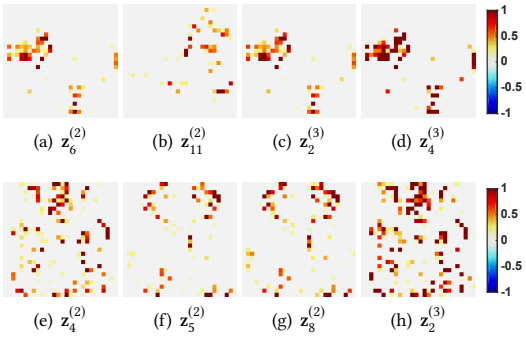
(a) Avg. Image (b) LR (c) LR-NS (d) PLNN (e) PLNN-NS

(f) Avg. Image (g) LR-F (h) LR-NSF (i) PLNN (j) PLNN-NS

Figure 5: The decision features of all models on FMNIST-2. (a)-(e) and (f)-(j) show the average image and the decision features of all models for *Coat* and *Pullover*, respectively. For PLNN and PLNN-NS, we show the decision features of the LLC whose convex polytope contains the most instances.

The superior accuracy of PLNN comes at the cost of cluttered decision features that may be hard to understand. Fortunately, applying non-negative and sparse constraints on PLNN effectively improves the interpretability of the decision features without affecting the classification accuracy.

In Figures 4 and 5, the decision features of PLNN-NS highlight similar image parts as LR-NS and LR-NSF, and are much easier to understand than the decision features of PLNN. In particular, in Figure 5, the decision features of PLNN-NS clearly highlight the collar and breast of *Coat*, and the shoulder of *Pullover*, which are much easier to understand than the cluttered features of PLNN. These results demonstrate the effectiveness of non-negative and sparse constraints in selecting meaningful features. Moreover, the decision features of PLNN-NS capture more details than LR-NS and LR-NSF, thus PLNN-NS achieves a comparable accuracy with PLNN, and significantly outperforms LR-NS and LR-NSF on FMNIST-2.



(a) $\mathbf{z}_6^{(2)}$ (b) $\mathbf{z}_{11}^{(2)}$ (c) $\mathbf{z}_2^{(3)}$ (d) $\mathbf{z}_4^{(3)}$

(e) $\mathbf{z}_4^{(2)}$ (f) $\mathbf{z}_5^{(2)}$ (g) $\mathbf{z}_8^{(2)}$ (h) $\mathbf{z}_2^{(3)}$

Figure 6: (a)-(d) show the PBFs of the PLNN-NS on FMNIST-1. (e)-(h) show the PBFs of the PLNN-NS on FMNIST-2.

Table 6: The PBs of the top-3 convex polytopes (CP) containing the most instances in FMNIST-1. "/" indicates a redundant linear inequality. Accuracy is the training accuracy of LLC on each CP.

CP	$\mathbf{z}_6^{(2)}$	$\mathbf{z}_{11}^{(2)}$	$\mathbf{z}_2^{(3)}$	$\mathbf{z}_4^{(3)}$	*#Ankle Boot*	*#Bag*	Accuracy
1	/	> 0	> 0	/	3,991	3,997	0.999
2	≤ 0	> 0	/	≤ 0	9	0	1.000
3	/	≤ 0	/	> 0	0	3	1.000

Table 7: The PBs of the top-3 convex polytopes (CP) containing the most instances in FMNIST-2. Accuracy is the training accuracy of LLC on each CP.

CP	$\mathbf{z}_4^{(2)}$	$\mathbf{z}_5^{(2)}$	$\mathbf{z}_8^{(2)}$	$\mathbf{z}_2^{(3)}$	*#Coat*	*#Pullover*	Accuracy
1	> 0	> 0	> 0	> 0	3,932	3,942	0.894
2	> 0	≤ 0	> 0	> 0	32	10	0.905
3	> 0	≤ 0	≤ 0	> 0	18	0	0.944

In summary, the decision features of LLCs are easy to understand, and the non-negative and sparse constraints are highly effective in improving the interpretability of the decision features of LLCs.

5.4 Are PBFs of LLCs Easy to Understand?

The **polytope boundary features (PBFs)** of **polytope boundaries (PBs)** interpret why an instance is contained in the convex polytope of a LLC. In this subsection, we study the semantical meaning of PBFs. Limited by space, we only use the PLNN-NS models trained on FMNIST-1 and FMINST-2 as the target model to interpret. The LLCs of PLNN-NS are computed by *OpenBox*.

Recall that a PB is defined by a linear inequality $\mathbf{z}_i^{(l)} \in \psi(\mathbf{c}_i^{(l)})$, where the PBFs are the coefficients of $\mathbf{x}$ in $\mathbf{z}_i^{(l)}$. Since the activation function is ReLU, $\mathbf{z}_i^{(l)} \in \psi(\mathbf{c}_i^{(l)})$ is either $\mathbf{z}_i^{(l)} > 0$ or $\mathbf{z}_i^{(l)} \leq 0$. Since the values of PBFs are non-negative for PLNN-NS, for a convex polytope P_h, if $\mathbf{z}_i^{(l)} > 0$, then the images in P_h strongly correlate with the PBFs of $\mathbf{z}_i^{(l)}$; if $\mathbf{z}_i^{(l)} \leq 0$, then the images in P_h are not strongly correlated with the PBFs of $\mathbf{z}_i^{(l)}$.

The above analysis of PBs and PBFs is demonstrated by the results in Tables 6 and 7, and Figure 6. Take the first convex polytope

in Table 6 as an example, the PBs are $\mathbf{z}_{11}^{(2)} > 0$ and $\mathbf{z}_{2}^{(3)} > 0$, whose PBFs in Figures 6(b)-(c) show the features of *Ankle Boot* and *Bag*, respectively. Therefore, the convex polytope contains images of both *Ankle Boot* and *Bag*. A careful study of the other results suggests that the PBFs of the convex polytopes are easy to understand and accurately describe the images in each convex polytope.

We can also see that the PBFs in Figure 6 look similar to the decision features of PLNN-NS in Figures 4 and 5. This shows the strong correlation between the features learned by different neurons of PLNN-NS, which is probably caused by the hierarchy network structure. Due to the strong correlation between neurons, the number of configurations in C is much less than k^N, as shown in Table 3.

Surprisingly, as shown in Table 7, the top-1 convex polytope on FMNIST-2 contains more than 98% of the training instances. On these instances, the training accuracy of LLC is much higher than the training accuracies of LR-NS and LR-NSF. This means that the training instances in the top-1 convex polytope are much easier to be linearly separated than all training instances in FMNIST-2. From this perspective, the behavior of PLNN-NS is like a "divide and conquer" strategy, which set aside a small proportion of instances that hinder the classification accuracy such that the majority of the instances can be better separated by a LLC. As shown by the top-2 and top-3 convex polytopes in Table 7, the set aside instances are grouped in their own convex polytopes, where the corresponding LLCs also achieve a very high accuracy. Table 6 shows similar phenomenon on FMNIST-1. However, since the instances in FMNIST-1 are easy to be linearly separated, the training accuracy of PLNN-NS marginally outperforms LR-NS and LR-NSF.

5.5 Can We Hack a Model Using OpenBox?

Knowing what an intelligent machine "thinks" provides us the privilege to "hack" it. Here, to hack a target model is to significantly change its prediction on an instance $\mathbf{x} \in \mathcal{X}$ by modifying as few features of $\mathbf{x}$ as possible. In general, the biggest change of prediction is achieved by modifying the most important decision features. A more precise interpretation on the target model reveals the important decision features more accurately, thus requires to modify less features to achieve a bigger change of prediction. Following this idea, we apply LIME and *OpenBox* to hack PLNN-NS, and compare the quality of their interpretations by comparing the change of PLNN-NS's prediction when modifying the same number of decision features.

For an instance $\mathbf{x} \in \mathcal{X}$, denote by $\gamma \in \mathbb{R}^d$ the decision features for the classification of $\mathbf{x}$. We hack PLNN-NS by setting the values of a few top-weighted decision features in $\mathbf{x}$ to zero, such that the prediction of PLNN-NS on $\mathbf{x}$ changes significantly. The change of prediction is evaluated by two measures as follows. First, the **change of prediction probability** (**CPP**) is the absolute change of the probability of classifying $\mathbf{x}$ as a positive instance. Second, the **number of label-changed instance** (**NLCI**) is the number of instances whose predicted label changes after being hacked. Again, due to the inefficiency of LIME, we use the sampled data sets in Section 5.2 for evaluation.

In Figure 7, the average CPP and NLCI of *OpenBox* are always higher than LIME on both data sets. This demonstrates that the

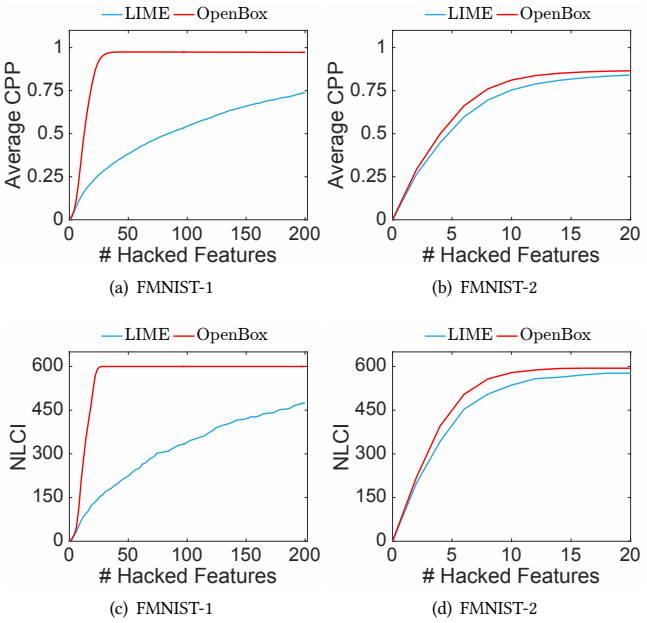


Figure 7: The hacking performance of LIME and *OpenBox*. (a)-(b) show the Average CPP. (c)-(d) show the NLCI.

interpretations computed by *OpenBox* are more effective than LIME when they are applied to hack the target model.

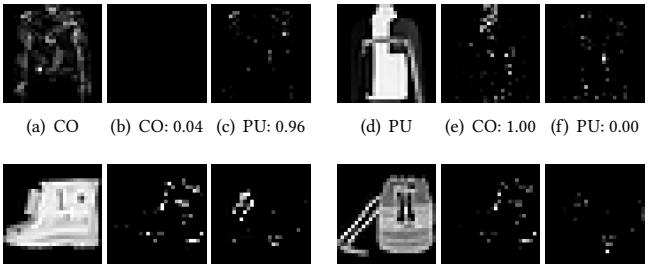
Interestingly, the advantage of *OpenBox* is more significant on FMNIST-1 than on FMNIST-2. This is because, as shown in Figure 2(a), the prediction probabilities of most instances in FMNIST-1 are either 1.0 or 0.0, which provides little gradient information for LIME to accurately approximate the classification function of the PLNN-NS. In this case, the decision features computed by LIME cannot describe the exact behavior of the target model.

In summary, since *OpenBox* produces the exact and consistent interpretations for a target model, it achieves an advanced hacking performance over LIME.

5.6 Can We Debug a Model Using OpenBox?

Intelligent machines are not perfect and predictions fail occasionally. When such failure occurs, we can apply *OpenBox* to interpret why an instance is mis-classified.

Figure 8 shows some images that are mis-classified by PLNN-NS with a high probability. In Figures 8(a)-(c), the original image is a *Coat*, however, since the scattered mosaic pattern on the cloth hits more features of *Pullover* than *Coat*, the original image is classified as a *Pullover* with a high probability. In Figures 8(d)-(f), the original image is a *Pullover*, however, it is mis-classified as a *Coat* because the white collar and breast hit the typical features of *Coat*, and the dark shoulder and sleeves miss the most significant features of *Pullover*. Similarly, the *Ankle Boot* in Figure 8(g) highlights more features on the upper left corner, thus it is mis-classified as a *Bag*. The *Bag* in Figure 8(j) is mis-classified as an *Ankle Boot* because it hits the features of ankle and heel of *Ankle Boot*, however, misses the typical features of *Bag* on the upper left corner.



(a) CO (b) CO: 0.04 (c) PU: 0.96 (d) PU (e) CO: 1.00 (f) PU: 0.00

(g) AB (h) AB: 0.16 (i) BG: 0.84 (j) BG (k) AB: 1.00 (l) BG: 0.00

Figure 8: The mis-classified images of (a) *Coat* (CO), (d) *Pullover* (PU), (g) *Ankle Boot* (AB), and (j) *Bag* (BG). (a), (d), (g) and (j) show the original images. For the rest subfigures, the caption shows the prediction probability of the corresponding class; the image shows the decision features supporting the prediction of the corresponding class.

In conclusion, as demonstrated by Figure 8, *OpenBox* accurately interprets the mis-classifications, which is potentially useful in debugging abnormal behaviors of the interpreted model.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we tackle the challenging problem of interpreting PLNNs. By studying the states of hidden neurons and the configuration of a PLNN, we prove that a PLNN is mathematically equivalent to a set of LLCs, which can be efficiently computed by the proposed *OpenBox* method. Extensive experiments show that the decision features and the polytope boundary features of LLCs provide exact and consistent interpretations on the overall behavior of a PLNN. Such interpretations are highly effective in hacking and debugging PLNN models. As future work, we will extend our work to interpret more general neural networks that adopt smooth activation functions, such as sigmoid and tanh.

REFERENCES

[1] Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. 2016. Analyzing the behavior of visual question answering models. *arXiv:1606.07356* (2016).
[2] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep?. In *NIPS*. 2654–2662.
[3] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. 2017. Interpreting Blackbox Models via Model Extraction. *arXiv:1705.08504* (2017).
[4] C Bishop. 2007. Pattern Recognition and Machine Learning (Information Science and Statistics). *Springer, New York* (2007).
[5] C. Cao, X. Liu, Y Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, et al. 2015. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. In *ICCV*. 2956–2964.
[6] RJ Caron, JF McDonald, and CM Ponic. 1989. A degenerate extreme point strategy for the classification of linear constraints as redundant or necessary. *JOTA* 62, 2 (1989), 225–237.
[7] Z. Che, S. Purushotham, R. Khemani, and Y. Liu. 2015. Distilling knowledge from deep networks with applications to healthcare domain. *arXiv:1512.03542* (2015).
[8] Jan Chorowski and Jacek M Zurada. 2015. Learning understandable neural networks with nonnegative weight constraints. *TNNLS* 26, 1 (2015), 62–69.
[9] Alexey Dosovitskiy and Thomas Brox. 2016. Inverting visual representations with convolutional networks. In *CVPR*. 4829–4837.
[10] D. Erhan, Yoshua Bengio, A. Courville, and P. Vincent. 2009. Visualizing higher-layer features of a deep network. *University of Montreal* 1341 (2009), 3.
[11] Ruth Fong and Andrea Vedaldi. 2017. Interpretable Explanations of Black Boxes by Meaningful Perturbation. *arXiv:1704.03296* (2017).
[12] Nicholas Frosst and Geoffrey Hinton. 2017. Distilling a Neural Network Into a Soft Decision Tree. *arXiv:1711.09784* (2017).
[13] Amirata Ghorbani, Abubakar Abid, and James Zou. 2017. Interpretation of Neural Networks is Fragile. *arXiv:1710.10547* (2017).
[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *ICAIS*. 315–323.
[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.
[16] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout networks. *arXiv:1302.4389* (2013).
[17] B. Goodman and S. Flaxman. 2016. European Union regulations on algorithmic decision-making and a" right to explanation". *arXiv:1606.08813* (2016).
[18] Nick Harvey, Chris Liaw, and Abbas Mehrabian. 2017. Nearly-tight VC-dimension bounds for piecewise linear neural networks. *arXiv:1703.02930* (2017).
[19] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*. 1026–1034.
[20] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv:1503.02531* (2015).
[21] Patrik O Hoyer. 2002. Non-negative sparse coding. In *WNNSP*. 557–565.
[22] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. 2017. The (Un) reliability of saliency methods. *arXiv:1711.00867* (2017).
[23] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. *arXiv:1703.04730* (2017).
[24] Pascal Koiran and Eduardo D Sontag. 1996. Neural networks with quadratic VC dimension. In *NIPS*. 197–203.
[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.
[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.
[27] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. 2007. Efficient sparse coding algorithms. In *NIPS*. 801–808.
[28] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2015. Visualizing and understanding neural models in NLP. *arXiv:1506.01066* (2015).
[29] Aravindh Mahendran and Andrea Vedaldi. 2015. Understanding deep image representations by inverting them. In *CVPR*. 5188–5196.
[30] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. On the number of linear regions of deep neural networks. In *NIPS*. 2924–2932.
[31] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*. 807–814.
[32] R. B. Palm. 2012. Prediction as a candidate for learning deep hierarchical models of data.
[33] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. 2013. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv:1312.6098* (2013).
[34] Nadeem N Rather, Chintan O Patel, and Sharib A Khan. 2017. Using Deep Learning Towards Biomedical Knowledge Discovery. *IJMSC* 3, 2 (2017), 1.
[35] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *KDD*. ACM, 1135–1144.
[36] R. R Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. 2016. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *arXiv:1610.02391* (2016).
[37] A. Shrikumar, P. Greenside, and A. Kundaje. 2017. Learning important features through propagating activation differences. *arXiv:1704.02685* (2017).
[38] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034* (2013).
[39] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. 2017. SmoothGrad: removing noise by adding noise. *arXiv:1706.03825* (2017).
[40] Eduardo D Sontag. 1998. VC dimension of neural networks. *NATO ASI Series F Computer and Systems Sciences* 168 (1998), 69–96.
[41] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. *arXiv:1703.01365* (2017).
[42] M. Wu, M. C Hughes, S. Parbhoo, M. Zazzi, V. Roth, and F. Doshi-Velez. 2018. Beyond Sparsity: Tree Regularization of Deep Models for Interpretability. *AAAI* (2018).
[43] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:cs.LG/cs.LG/1708.07747
[44] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. 2015. Understanding neural networks through deep visualization. *arXiv:1506.06579* (2015).
[45] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning fair representations. In *ICML*. 325–333.
[46] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. 2017. Interpreting Deep Visual Representations via Network Dissection. *arXiv:1711.05611* (2017).
[47] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *CVPR*. 2921–2929.
[48] J. Zhu, Y. Shan, JC Mao, D. Yu, H. Rahmanian, and Y. Zhang. 2017. Deep embedding forest: Forest-based serving with deep embedding features. In *KDD*. 1703–1711.