

Piecewise Linear Neural Networks and Deep Learning[#]

Qinghua Tao^{1,*}, Li Li^{2,*}, Xiaolin Huang^{3,*}, Xiangming Xi⁴, Shuning Wang², and Johan A.K. Suykens¹

¹STADIUS, ESAT, KU Leuven, Heverlee, 3001, Belgium

²Department of Automation, Tsinghua University, Beijing, 100084, China

³Department of Automation, Shanghai Jiao Tong University, Shanghai, 200240, China

⁴Zhejiang Lab, Hangzhou, 311121, China

*corresponding.author: qinghua.tao@esat.kuleuven.be, li-li@mail.tsinghua.edu.cn, xiaolinhuang@sjtu.edu.cn

[#] This work has been published in *Nature Reviews Methods Primers*. For citations, please cite “Tao, Q., Li, L., Huang, X. et al. Piecewise linear neural networks and deep learning. *Nature Reviews Methods Primers* 2, 42 (2022). <https://doi.org/10.1038/s43586-022-00125-7>”.

For the published version, please access <https://rdcu.be/cPIGw> for online-reading, download the manuscript from <https://www.nature.com/articles/s43586-022-00125-7#citeas>, and check the supplementary information via <https://www.nature.com/articles/s43586-022-00125-7#Sec38>.

ABSTRACT

As a powerful modelling method, Piecewise Linear Neural Networks (PWLNNs) have proven successful in various fields, most recently in deep learning. To apply PWLNN methods, both the representation and the learning have long been studied. In 1977, the canonical representation pioneered the works of shallow PWLNNs learned by incremental designs, but the applications to large-scale data were prohibited. In 2010, the Rectified Linear Unit (ReLU) advocated the prevalence of PWLNNs in deep learning. Ever since, PWLNNs have been successfully applied to extensive tasks and achieved advantageous performances. In this Primer, we systematically introduce the methodology of PWLNNs by grouping the works into shallow and deep networks. Firstly, different PWLNN representation models are constructed with elaborated examples. With PWLNNs, the evolution of learning algorithms for data is presented and fundamental theoretical analysis follows up for in-depth understandings. Then, representative applications are introduced together with discussions and outlooks.

1 Introduction

Piecewise Linear (PWL) Neural Networks (NN) have been studied particularly since the 1970s, and are now a successful mainstream method in deep learning. PWLNNs partition the domain into numerous sub-regions, each of which has localized linearity, whilst maintaining nonlinearity throughout the whole domain with great modelling flexibility¹. In PWLNNs, the PWL nonlinearity is realized through network architectures, which do not require explicit descriptions on sub-regions of localized linearity. PWLNNs are explicit in geometrical interpretation and flexible in approximation, such as an induced conclusion by the Stone-Weierstrass approximation theorem²[G]. Through PWLNN methods, complex nonlinear systems can be modelled by a finite number of linear functions localized over different sub-regions. This characteristic naturally bridges linearity and nonlinearity, and is more amenable to the modelling, learning, and analysis than other nonlinear methods.

PWLNNs are neural network-structured versions of PWL functions [G]. The conventional representation for PWL functions is given by a region-by-region manner, which explicitly lists each sub-region tessellating the domain and its localized

linear function³. The conventional representation demands an excessive number of parameters and is intractable in practical applications involving numerous sub-regions and complex domain configurations⁴. In contrast, PWLNNs efficiently represent PWL functions with compact expressions organized as network architectures, through which versatile functionalities can be facilitated. PWLNNs have now become a mainstream method for utilizing PWL functions in data science. To realize the power of PWLNNs in practice, two challenges need to be resolved: how to construct proper models to represent PWLNNs (representation) and how to effectively optimize the PWLNNs to well model the data or systems (learning). To this end, great efforts have been made by many researchers from different fields. These works can be categorized into two main groups: shallow (e.g., Canonical Piecewise Linear Representation (CPLR)⁵ [G]) and deep (e.g., networks with Rectified Linear Units (ReLU)⁶ [G]) PWLNNs. During the evolution from shallow to deep, the representation, learning and analysis of PWLNNs are closely related.

CPLR pioneered the compact expressions and analytical studies for PWL functions^{7,8}. Hinging hyperplanes⁹ [G] is another important representation model, constructed with geometrical explanations of hinges. These two compact representations result in PWLNNs with one hidden layer, namely

[§]The glossary items are marked with [G] and each of them is given with a succinct explanation in Section 8.

shallow PWLNNs, where absolute-value operators and maximization operators are adopted to induce PWL nonlinearity, as in ReLU for deep learning. Subsequent research has focused on variant representation models^{10–17}, theoretical analysis^{18–25}, learning algorithms^{26–31}, to name a few. Most of the resulting PWLNNs were shallow-architected, and yet their performances were limited in high-dimensional and large-scale problems. In 2010, ReLU successfully embedded PWL nonlinearity in the mainstream Deep neural networks (PWL-DNNs) and has achieved record-breaking performances in various benchmarks^{32–34}. For example, given the same number of neurons, PWL-DNNs have exponentially greater capacity than that of their shallow counterparts³⁵.

For learning algorithms, shallow PWLNNs commonly use incremental designs, which iteratively grow wider networks. PWL-DNNs inherit the regular learning of generic DNNs, in which network structures are predefined and parameters are optimized by the backpropagation strategy [G] and stochastic gradient descent (SGD) [G] algorithm. With the support of powerful graphical and tensor processing units and efficient implementation platforms such as PyTorch³⁶, the powerful learning ability of PWL-DNNs is fundamentally realized to tackle complex tasks, and PWL-DNNs have since become the mainstream deep learning method after decades of rigorous developments.

In this primer, the PWLNN method is systematically introduced and versatile perspectives are provided, to give a more insightful understanding towards representations, learning, and analysis. Some preliminaries are given, and details of how to construct representation models for PWLNNs (Experimentation) followed by learning algorithms for data analysis and their theoretical properties (Results). We introduce several representative applications of PWLNNs (Applications) and the use of PWLNNs for data under standard cases (Reproducibility and data deposition). To conclude, the ongoing issues (Limitations and optimizations) and some potential future directions (Outlook) are discussed.

2 Experimentation

In this section, some preliminaries are firstly given, and then the representations of PWLNNs are introduced in detail.

2.1 Preliminaries

2.1.1 PWL functions

Linear functions are basic mathematical models, but lack flexibility in practical scenarios which commonly pertain nonlinear natures. PWL functions are powerful remedies bridging linearity to nonlinearity for great model flexibility. PWL functions are not necessarily continuous; in practice, continuity pervasively and sometimes naturally exists. For the purpose of this Primer, we will discuss continuous PWL functions.

In the conventional representation, PWL functions are given by a region-by-region manner³. Let $f(\mathbf{x}) : \Omega \mapsto \mathbb{R}$ be a function defined in domain $\Omega \subseteq \mathbb{R}^n$. $f(\mathbf{x})$ is a PWL function, if it satisfies equations (1) and (2) (or (3)).

To be specific, the domain Ω is divided into a finite number of polyhedral sub-regions Ω_i with interior $\mathring{\Omega}_i$ satisfying

$$\bigcup \Omega_i = \Omega, \quad \mathring{\Omega}_{i_1} \cap \mathring{\Omega}_{i_2} = \emptyset, \quad \forall i_1 \neq i_2, \quad i_1, i_2 = 1, \dots, d, \quad (1)$$

by a finite set of boundaries $\mathcal{B} = \{\pi_j(\mathbf{x})\}_{j=1}^h$, such that each boundary is an $(n-1)$ -dimensional hyperplane characterized by $\pi_j(\mathbf{x}) := \boldsymbol{\alpha}_j^T \mathbf{x} - \beta_j = 0$ with $\boldsymbol{\alpha}_j \in \mathbb{R}^n$, $\beta_j \in \mathbb{R}$, and cannot be covered by any $(n-2)$ -dimensional hyperplane. There exists a finite number of linear functions $l_1(\mathbf{x}), \dots, l_d(\mathbf{x})$ which constitute the complete expression of $f(\mathbf{x})$, such that

$$f(\mathbf{x}) \in \{l_1(\mathbf{x}), \dots, l_d(\mathbf{x})\}, \quad \forall \mathbf{x} \in \Omega, \quad (2)$$

or equivalently

$$f(\mathbf{x}) = l_i(\mathbf{x}) = \mathbf{J}_i^T \mathbf{x} + b_i, \quad \forall \mathbf{x} \in \Omega_i, \quad i = 1, \dots, d, \quad (3)$$

where $\mathbf{J}_i \in \mathbb{R}^n$ is called the Jacobian vector of the sub-region Ω_i with the bias $b_i \in \mathbb{R}$.

For example, denoting $f : \mathbb{R}^3 \mapsto \mathbb{R}$ as a PWL function, by the conventional representation, $f(\mathbf{x})$ is expressed as

$$f(\mathbf{x}) = \begin{cases} l_1(\mathbf{x}) = x_1 - x_2 + x_3 + 1, & \pi(\mathbf{x}) \geq 0, \\ l_2(\mathbf{x}) = -x_1 + x_2 - x_3 - 1, & \pi(\mathbf{x}) < 0, \end{cases} \quad (4)$$

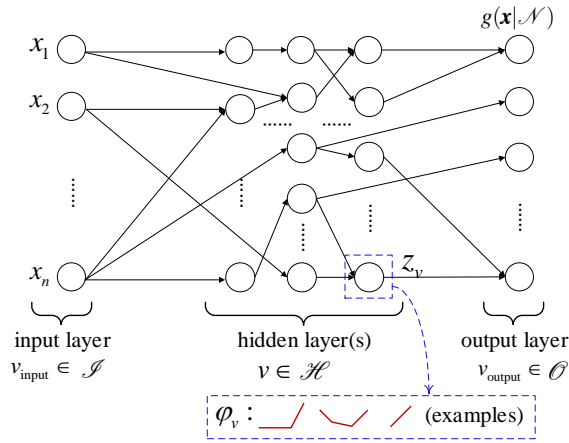
where the biases are $b_1 = 1$ and $b_2 = -1$, the boundary set \mathcal{H} contains $\pi(\mathbf{x}) := x_1 - x_2 + x_3 + 1 = 0$, the sub-regions are $\Omega_1 = \{\mathbf{x} \in \mathbb{R}^3 | x_1 - x_2 + x_3 + 1 \geq 0\}$ and $\Omega_2 = \{\mathbf{x} \in \mathbb{R}^3 | x_1 - x_2 + x_3 + 1 < 0\}$, and the Jacobian vectors of Ω_1 and Ω_2 are $\mathbf{J}_1 = [1, -1, 1]^T$ and $\mathbf{J}_2 = [-1, 1, -1]^T$, respectively.

Note that with $d = 1$, the PWL function $f(\mathbf{x})$ is reduced to be linear, thus, we regard linear functions as a special case of PWL functions throughout the Primer. Compared to other nonlinear models, PWL functions possess explicit geometric interpretation, and many practical systems can be easily transformed into PWL nonlinear functions³⁷, such as PWL memristors [G]^{38,39}, specialized cost functions^{40–44}, and part mathematical programmings^{45–50}. As powerful nonlinear models, PWL functions are proven universal approximators⁵¹: let $\Omega \subset \mathbb{R}^n$ be a compact domain, and $p(\mathbf{x}) : \Omega \mapsto \mathbb{R}$ be a continuous function. When $\forall \varepsilon > 0$, there exists a PWL function $f(\mathbf{x})$ (depending upon ε), such that $\forall \mathbf{x} \in \Omega, |f(\mathbf{x}) - p(\mathbf{x})| < \varepsilon$.

2.1.2 PWLNNs

Before applying the PWL nonlinearity, it is important to construct proper mathematical formulas to represent such PWL functions. Neural networks have been widely recognized as one of the most powerful nonlinear approximators in data science. Neural networks refer to mathematical layered models composed of artificial neurons and their connections⁵². Mapping through neurons introduces nonlinearity by activation functions, leading to powerful flexible models. In neural networks, such mappings are realized via the mapping functions of neurons. In PWLNNs, such nonlinear mapping functions are specified as PWL functions, bringing PWL nonlinearity to the network. Box 1 defines PWLNNs from the perspective of

Box 1 | PieceWise Linear Neural Networks (PWLNNs)



An illustration of PWLNNs defined below.

Let $\mathbf{g}(\mathbf{x}|\mathcal{N})$ be a PWLNN with the architecture $\mathcal{N} = (\mathcal{V}, \mathcal{E})$, determined by a finite set of neurons, \mathcal{V} , and a finite set of edges, \mathcal{E} , depicting the directed connections between paired neurons in \mathcal{V} . Given an edge $e = (v', v) \in \mathcal{E}$, it is an incoming edge for v and outgoing edge for v' . Neurons in \mathcal{V} can be grouped into non-overlapping and non-empty subsets consisting of input neurons (\mathcal{I}), output neurons (\mathcal{O}), and hidden neurons (\mathcal{H}), such that $\mathcal{V} = \mathcal{I} \cup \mathcal{O} \cup \mathcal{H}$.

The architecture of such a network can be structured into layers. Neurons in \mathcal{I} constitute the input layer, or the 0-th layer. Neurons in \mathcal{H} can be organized into hidden layers L ($L \geq 1$) connected sequentially, such that a neuron in the l -th ($1 \leq l \leq L$) layer only has incoming edges from neurons in precedent layers (including the input layer) and has outgoing edges to neurons in subsequent layers (including the output layer). The output layer is constructed by neurons in \mathcal{O} , and is also called the $(L+1)$ -th layer.

For any neuron $v \in \mathcal{V} \setminus \mathcal{I}$, there is either a linear or PWL mapping function $\phi_v : \mathbb{R}^{n_v} \rightarrow \mathbb{R}$, so that

$$z_v = \phi_v(z_{v_1}, \dots, z_{v_{n_v}}; \boldsymbol{\theta}_v)$$

where z_v is the neuron output, n_v is the number of incoming edges for v , which are directed from $v_i, i = 1, \dots, n_v$ in precedent layers to v , and $\boldsymbol{\theta}_v$ denotes the learnable parameters related to neuron v . With $L = 1$, it is a shallow PWLNN, when $L \geq 2$ it is a PWL-DNN.

network architectures. This definition has been generalised to include the interconnection weight of an edge (v', v) between neuron v' and neuron v , as generally used in neural networks, into the PWL mapping function of the neuron v . This makes PWLNNs more expressible for complex mapping functions in

neurons, such as those used in the Maxout neural networks⁵¹. Furthermore, all the parameters $\Theta = \{\boldsymbol{\theta}_v\}_{v \in \mathcal{V} \setminus \mathcal{I}}$ constitute the parameter space of the network, and correspond to the commonly-used interconnection weights and bias terms in other descriptions for neural networks.

Two main aspects should be considered before applying PWLNNs in practice. One is the determination of the network structures of \mathcal{N} , and the other is the determination of the learnable parameters.

PWLNNs are the network-structured versions of PWL functions. However, given a specific representation model of PWLNNs, where \mathcal{N} is specified to a certain class of network structures, this representation model does not necessarily have universal representation ability for arbitrary PWL functions in \mathbb{R}^n . This leads to the definition of universal representation ability as follows: when \mathcal{F} is the set of all PWL functions as defined by the conventional representation, a PWLNN representation model $\mathbf{g}(\mathbf{x}|\mathcal{N})$ with a single output neuron in \mathcal{O} is said to have universal representation ability for \mathcal{F} , if the following condition is satisfied

$$\forall f \in \mathcal{F}, \exists \mathcal{N} \text{ such that } f(\mathbf{x}) = \mathbf{g}(\mathbf{x}|\mathcal{N}). \quad (5)$$

Besides the determination of the network architecture of a PWLNN, how to effectively identify model parameters is also critical to attain good performance in practice. Figure 1 gives the workflow when applying PWLNNs to specific tasks.

2.2 Shallow PWLNNs

In shallow PWLNNs, there are two main types of representations: the models consisting of basis functions and the Lattice representation. An overall sketch on the chronicle of all surveyed PWLNN representations can be found in the supplementary information.

2.2.1 Representations based on basis functions – the canonical family

The representation models consisting of basis functions are formulated in the form of

$$f(\mathbf{x}) = \sum_{m=1}^M w_m B_m(\mathbf{x}; \boldsymbol{\theta}_{v_m}), \quad (6)$$

where $B_m(\mathbf{x}; \boldsymbol{\theta}_{v_m})$ is the m -th basis function introducing PWL nonlinearity, w_m is the coefficient, and M is the number of basis functions. The general network architecture \mathcal{N} resulting from equation (6) is shown in Figure 2(a), where one hidden layer is utilized. In such PWLNNs, the PWL mapping function ϕ_{v_m} of each hidden neuron $v_m, m = 1, \dots, M$, corresponds to each basis function $B_m(\mathbf{x}; \boldsymbol{\theta}_{v_m})$, which can be with learnable parameters $\boldsymbol{\theta}_{v_m}$, where the incoming edges of each hidden neuron are from the n neurons in the input layer. The PWL mapping function of the neuron v_{output} in the output layer is the weighted sum of the values from its incoming edges, where the weights correspond to $\boldsymbol{\theta}_{v_{\text{output}}} = \{w_m\}_{m=1, \dots, M}$. In this type of PWLNN representations, the canonical and the hinge-based families are mainly involved.

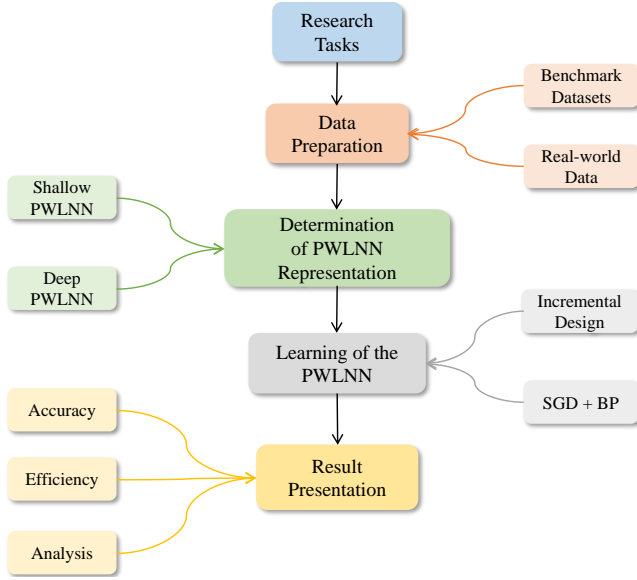


Figure 1. A general workflow of applying the PWLNN method. Given a task of prediction or analysis, standard procedures start from data preparation, where pre-processing or data-augmentation techniques can be involved. Then, an appropriate PWLNN representation is selected to perform the modelling together with a learning algorithm for fitting the data, where the selection can be determined by practitioners' particular interest or through trial-and-error strategy. With the learned PWLNN, prediction outputs of the given task can be computed, where empirical and theoretical investigations can both be conducted to present the final results.

CPLR was originally proposed in the univariate formulation⁵ and has been extended to higher dimensions⁷ by

$$f(\mathbf{x}) = \boldsymbol{\alpha}_0^T \mathbf{x} + \beta_0 + \sum_{m=1}^M \eta_m |\boldsymbol{\alpha}_m^T \mathbf{x} + \beta_m|, \quad (7)$$

where $\mathbf{x} \in \mathbb{R}^n$ is the input vector, $\eta_m = \pm 1$, $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_m \in \mathbb{R}^n$ and $\beta_0, \beta_m \in \mathbb{R}$ are the parameters. Figure 3(a) gives the plot of a simple PWLNN for an illustration on the CPLR representation. For example, given a univariate PWL function $f(x)$

$$f(x) = \begin{cases} x+2, & x \in (-\infty, -1], \\ -x, & x \in (-1, 1], \\ x-2, & x \in (1, \infty], \end{cases} \quad (8)$$

it can be represented by CPLR as follows

$$f(x) = x - |x+1| + |x-1|, \quad (9)$$

with three basis functions, each of which corresponds to the PWL mapping function in each of the resulting hidden neurons. As indicated in Figure 2(a), the output neuron v_{output} has incoming edges from three hidden neurons in $\mathcal{H} = \{v_1, v_2, v_3\}$, and the output neuron's output is the weighted

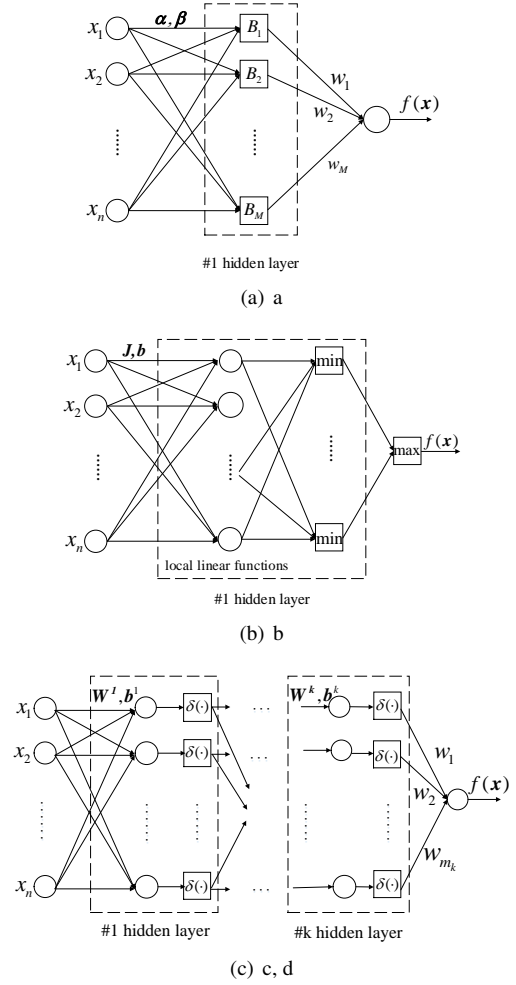


Figure 2. An illustration on the topology of PieceWise Linear Neural Network (PWLNN) representations, where the outputs of squared nodes denote the PWL mappings. a) Representation of basis functions in equation (6); b) Lattice representation of equation (20); c) PWL-DNNs of equation (22) with 1 hidden layer; d) PWL-DNNs of equation (22) with k hidden layers.

sum of the hidden neurons' outputs from its three incoming edges, where the weights are $\boldsymbol{\theta}_{v_{\text{output}}} = \{1, -1, 1\}$. Based on Box 1, the network structure of this PWLNN is thereby built upon the neurons $\mathcal{V} = \{v_{\text{input}}, v_1, v_2, v_3, v_{\text{output}}\}$ and the edges $\mathcal{E} = \{(v_{\text{input}}, v_1), (v_{\text{input}}, v_2), (v_{\text{input}}, v_3), (v_1, v_{\text{output}}), (v_2, v_{\text{output}}), (v_3, v_{\text{output}})\}$, as illustrated in Figure 3(c).

We can also remove v_1 from \mathcal{H} and build a PWLNN with only two hidden neurons if we replace the edges with $(v_{\text{input}}, v_{\text{output}})$ in a skip-layer manner (Figure 3(d)). It shows that a PWL function can be formulated into varied PWLNNs with different network structures.

However, a crucial problem exists: CPLR can represent arbitrary PWL functions in \mathbb{R} but this representation ability is harmed in $\mathbb{R}^n (n \geq 2)$ ⁴. For example¹⁴, given a PWL function

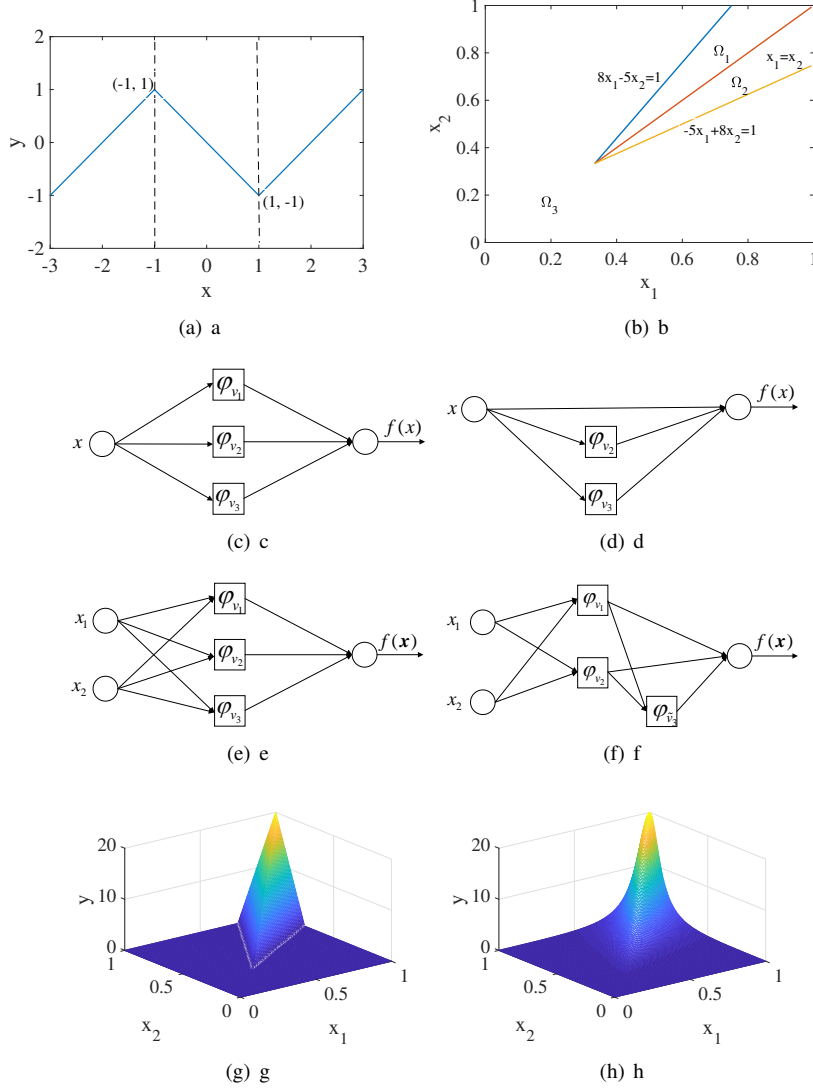


Figure 3. Simple illustrations to visualise the resulting PWLNNs in equations (9) and (12). a) Plot of equation (9) to illustrate the canonical piecewise linear representation (CPLR) representation; b) Domain configuration of equation (12); c, d) shallow PWLNNs for (9); e) a shallow PWLNN for equation (12); f) a two-hidden-layered PWLNN for equation (12); g) Plot of equation (12); h) An exemplified function to approximate when considering using equation (12).

$f: \mathbb{R}^2 \mapsto \mathbb{R}$ with $\mathbf{x} = [x_1, x_2]^T$ as follows

$$f(\mathbf{x}) = \begin{cases} l_1(\mathbf{x}) = 80x_1 - 50x_2 - 10 & \mathbf{x} \in \Omega_1, \\ l_2(\mathbf{x}) = -50x_1 + 80x_2 - 10 & \mathbf{x} \in \Omega_2, \\ l_3(\mathbf{x}) = 0 & \mathbf{x} \in \Omega_3, \end{cases} \quad (10)$$

its boundaries contain the ridges $x_1 = x_2$, $8x_1 - 5x_2 = 1$ and $-5x_1 + 8x_2 = 1$. However, these ridges vanish in the sub-region $\Omega_3 = \{\mathbf{x} \mid 3x_1 \leq 1, 2x_2 \leq 1\}$, and such domain configurations cannot be realized by any CPLR. This incomplete representation ability can essentially hurt the approximation performance when applying PWLNNs to data in the CPLR representation. For example, Figure 3(g) can be an approximator for the function in Figure 3(h), but it cannot be repre-

sented in the form of CPLR. Note that although the simple PWL approximator in Figure 3(g) cannot be represented by CPLR, there exist other PWLNNs in the form of CPLR that can approximate Figure 3(h) to arbitrary accuracy, due to its universal approximation ability⁴.

To overcome the incomplete representation ability of CPLR in \mathbb{R}^2 , the two-level nesting of CPLR is constructed⁵³:

$$f(\mathbf{x}) = \boldsymbol{\alpha}_0^T \mathbf{x} + \beta_0 + \sum_{m=1}^M \eta_m |\boldsymbol{\alpha}_{m,1}^T \mathbf{x} + \beta_{m,1} + |\boldsymbol{\alpha}_{m,2}^T \mathbf{x} + \beta_{m,2}||. \quad (11)$$

Then, equation (10) can be represented as

$$f(\mathbf{x}) = 7.5(x_1 + x_2) - 5 - 32.5|x_1 - x_2| + |32.5|x_1 - x_2| - 7.5(x_1 + x_2) + 5|. \quad (12)$$

Similar to equation (9), the resulting PWLNN of equation (12) has three hidden neurons with PWL mapping functions corresponding to basis functions. The neurons $\mathcal{N} = \{v_{\text{input},1}, v_{\text{input},2}, v_1, v_2, v_3, v_{\text{output}}\}$ and the edges $\mathcal{E} = \{(v_{\text{input},1}, v_1), (v_{\text{input},2}, v_1), (v_{\text{input},1}, v_2), (v_{\text{input},2}, v_2), (v_{\text{input},1}, v_3), (v_{\text{input},2}, v_3), (v_1, v_{\text{output}}), (v_2, v_{\text{output}}), (v_3, v_{\text{output}})\}$ are obtained, as shown in Figure 3(e).

It is worth mentioning that different network structures can also be built for equation (12) and it is of particular interest for v_3 having a PWL mapping function based on the two-level nesting of absolute-value operators. Figure 3(f) gives an illustration of another PWLNN variant by reformulating v_3 into \tilde{v}_3 . This means equation (12) can also be interpreted as a deep architecture with two hidden layers, where the original PWL mapping function $|32.5|x_1 - x_2| - 7.5(x_1 + x_2) + 5|$ is equivalently formulated into a simplified form of $|-z_{v_2} - z_{v_1}|$ by transforming the incoming edges $(v_{\text{input},1}, v_3)$ and $(v_{\text{input},2}, v_3)$ from input neurons into (v_1, \tilde{v}_3) and (v_2, \tilde{v}_3) from the 1st hidden layer. v_1 can also be reformulated with skip-layer connections to the output neuron as done in Figure 3(d), but the details are not exhaustively presented here.

Nonetheless, in \mathbb{R}^n ($n > 2$), the two-level nesting of CPLR cannot cover all PWL functions⁵³. Thus, more flexible representations are needed. Similar to the two-level nesting, Generalized CPLR (G-CPLR) can be used⁵⁴. G-CPLR refers to any finite number of nestings of CPLR (defined as K -level CPLR with $K \geq 1$), which takes the form

$$f(\mathbf{x}) = f_0(\mathbf{x}) + C|f_1(\mathbf{x})|, \quad (13)$$

where both $f_0(\mathbf{x})$ and $f_1(\mathbf{x})$ are models of up to $K - 1$ levels of nested CPLR. Theoretically, G-CPLR can express all PWL functions in n dimensions with at most n -level CPLR⁵⁴.

With G-CPLR, the canonical formulations in equations (7) and (11) can be regarded as the one-level CPLR and the two-level CPLR, respectively. More generally than Figure 3(f), G-CPLR can be regarded as a trial for deep-architected PWLNNs. However, G-CPLR was mainly of theoretical significance for universal representation ability, and an effective learning algorithm has yet to be constructed.

2.2.2 Representations based on basis functions – the hinge family

The hinge family refers to the PWLNN representations based on hinge functions where two hyperplanes continuously join together (Figure 4).

Given two hyperplanes $h^+ = \boldsymbol{\alpha}_+^T \mathbf{x} + \beta_+$ and $h^- = \boldsymbol{\alpha}_-^T \mathbf{x} + \beta_-$ joining together at $\{\mathbf{x} | (\boldsymbol{\alpha}_+ - \boldsymbol{\alpha}_-)^T \mathbf{x} = 0\}$, their joint is defined as the hinge for h^+ and h^- and is formulated as $\max\{h^+, h^-\}$. The model of hinging hyperplanes⁹ is given by $f(\mathbf{x}) = \sum_{m=1}^M w_m \max\{\boldsymbol{\alpha}_{m+}^T \mathbf{x} + \beta_{m+}, \boldsymbol{\alpha}_{m-}^T \mathbf{x} + \beta_{m-}\}$, commonly

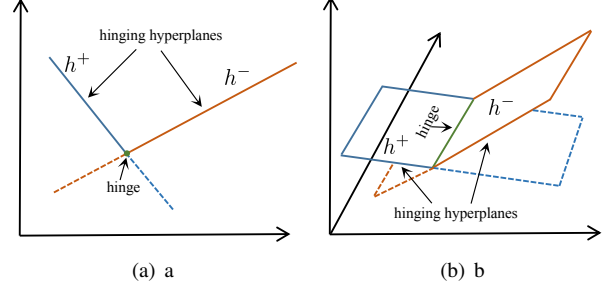


Figure 4. An illustration on the hinge function and its hinging hyperplanes. a) One-dimensional hinge; b) Two-dimensional hinge.

used as

$$f(\mathbf{x}) = \boldsymbol{\alpha}_0^T \mathbf{x} + \beta_0 + \sum_{m=1}^M w_m \max\{\boldsymbol{\alpha}_m^T \mathbf{x} + \beta_m, 0\}. \quad (14)$$

Here, the hinges have more distinguishable geometrical explanations than that of CPLR, making it more amenable to design effective learning algorithms^{55–59}.

Similar to CPLR, the hinging hyperplanes still cannot represent all PWL functions in \mathbb{R}^n ($n \geq 2$). To amend the incomplete representation ability, the Generalized hinging hyperplanes (GHH)¹⁴ can be adopted by adding a sufficient number of linear functions to the hinges, such that

$$f(\mathbf{x}) = \sum_{m=1}^M w_m \max\{\boldsymbol{\alpha}_1^T \mathbf{x} + \beta_1, \dots, \boldsymbol{\alpha}_{k_m+1}^T \mathbf{x} + \beta_{k_m+1}\}. \quad (15)$$

Given that $k_m \leq n$, this is also referred to as the n -order Hinging Hyperplanes (n -HH); any n -dimensional PWL function can be represented by an n -HH¹⁴.

Similarly, equation (10) can be represented in the GHH form:

$$f(\mathbf{x}) = \max\{65(x_1 - x_2), 65(x_2 - x_1), 15(x_1 + x_2) - 10\} - \max\{65(x_1 - x_2), 65(x_2 - x_1)\}, \quad (16)$$

which results in two hidden neurons with the PWL mappings based on 2-HH. The output neuron gives a weighted sum of the neuron outputs from the hidden layer with weights 1 and -1 . In the form of GHH, equation (10) can be represented by a PWLNN with only two hidden neurons, resulting in simpler network structures but requiring more flexible PWL mapping functions in the hidden layer.

2.2.3 Representations based on basis functions – others

There are other basis functions, which are designed with joint considerations to flexibility and learnability. Based on simplicial partitions and vertex interpolation, High-Level CPLR (HL-CPLR)⁶⁰ can be constructed with the basis functions as

$$B_{j_{k_1}, \dots, j_{k_R}}^R(\mathbf{x}) = \max\{0, \min\{x_{k_1} - j_{k_1}d, \dots, x_{k_r} - j_{k_r}d, \dots, x_{k_R} - j_{k_R}d\}\},$$

(17)

where d is the partitioning interval, $k_r \in \{1, \dots, n\}$ varies with $r = 1, \dots, R$, and j_{k_r} is selected from the $\{1, \dots, m_{k_r}\}$ partitions on the axis. Similar ideas of vertex-based modelling can also be found elsewhere^{61–64}.

By utilizing recursive domain partitions, the Adaptive Hinging Hyperplanes (AHH)¹⁶ is formulated as:

$$f(\mathbf{x}) = \sum_{m=1}^M w_m \min_j \{\max\{0, \delta_{j,m}(x_{v_{j,m}} - \beta_{j,m})\}\}, \quad (18)$$

where $x_{v_{j,m}}$ is the $v_{j,m}$ -th input variable, and $\beta_{j,m}$ is the splitting knot, $j \in J_m$, $J_m \subseteq \{1, \dots, n\}$ is the set containing the indices of input variables involved in the m -th basis function, and $\delta_{j,m} = \pm 1$. AHH can be regarded as a special case of GHH, and HL-CPLR is a special case of AHH.

The Simplex Basis Function (SBF)¹⁷ model is given by

$$f(\mathbf{x}) = \sum_m w_m \max\{0, 1 - \sum_{i=1}^n \gamma_{m,i} |x_i - \zeta_{m,i}|\}, \quad (19)$$

where $\gamma_{m,i}$ and $\zeta_{m,i}$ are the parameters controlling the shape of the m -th basis function. For any SBF, equivalent transformations to HH and CPLR also exist.

2.2.4 Lattice Representations

The Lattice representation⁶⁵ is constructed based on the Lattice theory⁶⁶, where simply the “max-min” composition of linear functions is sufficient¹¹. It is worth mentioning that the “max-min” and the absolute-value operators for PWL nonlinearity have been addressed in mathematical programming and functional analysis. For example, major attention is given to aspects of Lipschitz continuity, nondifferentiability, nonsmoothness and their algorithmic aspects, rather than to PWLNNs and data driven applications^{67–70}.

Letting $f(\mathbf{x})$ be an arbitrary PWLNN with d distinct linear functions, the Lattice representation is formulated as

$$f(\mathbf{x}) = \max_{i \in \mathbf{Z}(M)} \min_{j \in S_i} \{\mathbf{J}_j^T \mathbf{x} + b_j\}, \quad (20)$$

where $\mathbf{Z}(M) = \{1, 2, \dots, M\}$, $[\mathbf{J}_1, \dots, \mathbf{J}_d] \in \mathbb{R}^{n \times d}$, $[b_1, \dots, b_d]^T \in \mathbb{R}^d$ and $S_i \subseteq \mathbf{Z}(d)$. Equation (21) gives a simple illustration of the Lattice representation model. Given a PWL function $f: [0, 5] \mapsto \mathbb{R}$

$$f(x) = \begin{cases} l_1(x) = 0.5x + 0.5, & x \in \Omega_1 = [0, 1], \\ l_2(x) = 2x - 1 & x \in \Omega_2 = [1, 1.8], \\ l_3(x) = 2 & x \in \Omega_3 = [1.8, 3.2], \\ l_4(x) = -2x + 9 & x \in \Omega_4 = [3.2, 4], \\ l_5(x) = -0.5x + 3 & x \in \Omega_5 = [4, 5], \end{cases} \quad (21)$$

its Lattice representation model is formulated as $M = 5$, $S_1 = \{1, 3, 4, 5\}$, $S_2 = \{2, 3, 4, 5\}$, $S_3 = \{2, 3, 4\}$, $S_4 = \{1, 2, 3, 4\}$ and $S_5 = \{1, 2, 3, 5\}$. The resulting PWLNN based on the Lattice representation has 5 hidden neurons with PWL mapping functions based on the “min” operators across a maximum of

5 linear functions (Figure 2(b)). The PWL mapping function of the output neuron is simply the “max” operator across a maximum of 5 outputs of hidden neurons (M). Although the Lattice representation is formulated quite differently from those based on basis functions, it plays a significant role in the theoretical analysis of different PWLNN representations to reveal their intrinsic relations and modelling evolution.

Given a PWL function, PWLNNs with different network structures \mathcal{V} and edges \mathcal{E} can be constructed, when different PWL mapping functions are chosen for neurons $\mathcal{V} \setminus \mathcal{S}$. Such PWL mapping functions are built upon either the “min”, the “max”, the absolute-value operators, or their multi-level nestings. Often, more flexible PWL mapping functions result in less neurons and edges, while with simpler PWL mapping functions the network can be possibly reorganized into more layers, for example the cases shown in Figure 3.

2.3 PWL-DNNs

Mainstream DNNs are sequentially composed of an input layer, multiple hidden layers and an output layer (Figure 2(c)). Through the incoming edges, the outputs of neurons in the previous layer are linearly weighted and summed. Upon application of an activation function, nonlinearity is induced. Generally, the outputs of neurons in the k -th ($k \geq 1$) layer in DNNs is computed as

$$\mathbf{f}^k(\mathbf{x}) = \sigma^k(\Xi^k(\mathbf{x})), \quad (22)$$

where $\Xi^k(\mathbf{x}) = \mathbf{W}^k \mathbf{f}^{k-1}(\mathbf{x}) + \mathbf{b}^k$ denotes the weighted sum. Commonly in deep learning, $\mathbf{W}^k \in \mathbb{R}^{m_k \times m_{k-1}}$ and $\mathbf{b}^k \in \mathbb{R}^{m_k}$ are the weight matrix and the bias term for this neuron, respectively, where m_k and m_{k-1} refer to the numbers of neurons in the k -th and the $(k-1)$ -th layers, respectively. $\sigma^k(\cdot)$ is the nonlinear activation function, usually in a very simple form. In DNNs, the idea is to introduce PWL activation functions $\sigma^k(\cdot)$ to neurons in the hidden layers, so that PWL feature mappings are composited across layers for greater flexibility. Following notations in Box 1, the network architecture $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ of equation (22) confines neurons in a hidden layer to only have incoming edges from those in the previous layer, and the corresponding PWL mapping functions take the form of $\phi(\mathbf{f}^{k-1}; \mathbf{W}^k, \mathbf{b}^k) = \sigma^k(\mathbf{W}^k \mathbf{f}^{k-1} + \mathbf{b}^k)$. They take the parameters in \mathbf{W}^k as weights assigned to incoming edges of neurons, and \mathbf{b}^k as a bias term to the weighted sum of the outputs of neurons in the previous layer.

It is demonstrated that ReLU can significantly alleviate the gradient vanishing problem [G] in DNNs⁷¹. The recent prevalence of ReLU in deep learning^{32,33} showcases the great flexibility and power of PWL-DNNs in various complex tasks, though the very first PWL neuron stemmed from the modified threshold logic unit early in 1940s⁷² and multi-layer PWLNNs were discussed in 1990s^{54,73,74}. PWL activation functions, such as ReLU, are now acknowledged as the first choice in deep learning, owing to their advantageous performances such as better generalization performance and faster computation⁷⁵. In order to enhance network flexibility and diversity, multiple variants of PWL activations have been proposed in deep

learning, mostly by re-shaping the hinge of ReLU. In fact, the resulting PWL mapping functions of the neurons deploying variant activations resemble the basis functions in the shallow PWLNN architectures. For instance, a shallow neural network with ReLU in equation (22) is equivalent to the model of hinging hyperplanes, or can be equivalently transformed into CPLR and SBF. Another typical example is the neural networks with Maxout⁵¹, which resemble GHH¹⁴. Table 1 summarizes the PWL activation functions in PWL-DNNs and their shallow-architected counterparts.

As indicated in Box 1, PWLNNs can be flexibly designed in many ways. Firstly, with the edges in \mathcal{E} , the connections between neurons in \mathcal{V} can be formulated in different ways through the mappings (shallow PWLNN representations), rather than simply the linearly weighted sum on the previous layer (deep learning). Skip-layer connections are also naturally allowed for any paired neurons. Skip-layer connections have been widely applied to shallow PWLNN representation models, such as the PWLNN in Figure 3(f) and the variant of AHH⁸⁵, which enables sparse and even decomposable network structures with great interpretability. In recent years, the flexible skip-layer connections have also received wide attention in deep learning, such as the ResNet³³ and its generalization, the DenseNet³⁴. Secondly, the mapping function ϕ_v can be multi-dimensional, meaning that each incoming edge of v can be assigned weights in vectors, such as the PWL-DNNs with Maxout and shallow PWLNNs with GHH and AHH. The activation functions in PWL-DNNs can also include learnable parameters for more flexible PWL mappings for each neuron, such as in S-shaped ReLU⁸¹ and APL⁸³, where multiple breakpoints exist and can be made learnable. Similar ideas were realised early in the 1990s⁸⁶.

Figure 2 compares the topology of PWLNNs. With limited computational resources, shallow architectures are usually the better choice for small-scale problems, owing to their flexibility and the alleviation of overfitting and model redundancy issues. For example, SBF is more efficient in its learning scheme and shows to be more robust against noises, whilst AHH and the Lattice have better interpretability. In particular, when the contributions of input variables or the effects of variable connections need to be explored, AHH is a good predictor. When local explicitness is required, the Lattice is the preferred choice, as in the explicit Model Predictive Control (MPC) problems. When tremendous data are given and highly complicated tasks are involved, PWL-DNNs are the superior choice, owing to their optimization techniques and mature computational platforms. Though no single model is capable of resolving all problems, there are a number of well-designed techniques in PWLNNs available for different demands. PWLNNs are natural embodiments of PWL functions exerting network structures, so that the powerful learning ability and other merits of (D)NNs can be implemented. At the same time, the introduction of PWL nonlinearity helps resolve some learning barriers in deep learning with significant developments. PWLNNs and deep learning have boosted

each other, together bringing unprecedented success in the big data era.

3 Results

Learning algorithms are the key to applying PWLNN models to practical tasks. This includes how to effectively determine the network parameters and structures to well resolve the tasks. A chronicle overview on learning algorithms can be found in the supplementary information. In this section, we look into PWLNN models and their learning with theoretical analysis to better understand the mechanism.

3.1 Learning shallow PWLNNs

Shallow PWLNNs are commonly learned incrementally. Computational platforms and hardware were limited in the 1970s-2000s, and so the incremental design was a good choice to balance efficiency and accuracy. These incremental learning algorithms differ depending on the representations; they can be categorized into the hinge finding algorithm, the tree searching algorithm, the structured decision algorithm, and others.

3.1.1 Hinge finding algorithm and Newton's algorithm

To learn the hinging hyperplanes, new basis functions are incrementally added in each (M -th) iteration by adopting the hinge finding algorithm on $y = f(\mathbf{x}) - \sum_{m=1}^{M-1} w_m B(\mathbf{x})$ and adjusting the sum $\sum_{m=1}^M w_m B(\mathbf{x})$. The key step in hinge finding is to perform the least squares method [G] to determine the estimated parameters of the hyperplanes fitted to S_- and S_+ :

$$\begin{aligned}\alpha_{m+} &= \left(\sum_{\mathbf{x}^i \in S_+} \mathbf{x}^i \mathbf{x}^{iT} \right)^{-1} \sum_{\mathbf{x}^i \in S_+} \mathbf{x}^i y^i, \\ \alpha_{m-} &= \left(\sum_{\mathbf{x}^i \in S_-} \mathbf{x}^i \mathbf{x}^{iT} \right)^{-1} \sum_{\mathbf{x}^i \in S_-} \mathbf{x}^i y^i,\end{aligned}\quad (23)$$

where $S_+ = \{\mathbf{x} : \mathbf{x}^T (\alpha_{m+} - \alpha_{m-}) > 0\}$ and $S_- = \{\mathbf{x} : \mathbf{x}^T (\alpha_{m+} - \alpha_{m-}) \leq 0\}$.

In this way, new basis functions are incrementally obtained by locating the hinge functions over the training data. In fact, the estimation in the hinge finding is a special case of the Gauss-Newton algorithm [G]⁵⁶. Similarly, the learning of GHH can inherit such algorithm, where multiple linear functions are involved in each hinge²⁹. Simultaneously identifying all basis functions is also possible⁵⁶, but whether the sequential or simultaneous update of basis functions is the better option, if the computational resources are adequate, remains an open question.

3.1.2 Tree Searching Algorithm

AHH can be seen as a PWL analogy to the multivariate adaptive regression splines⁸⁷ [G]. Similarly, the learning of AHH is based on the recursive domain partitions by deploying the tree searching algorithm; the learning process of AHH can be interpreted as a generic tree with basis functions $B_m(\mathbf{x})$ as leaf nodes (Figure 5). Each basis function (neuron) $B_m(\mathbf{x})$ in Iterations 1-4 correspond to a sub-region T_m , giving the resulting tree topology of learning AHH (Figure 5(e)). The

Table 1. Descriptions on the surveyed PWL activation functions in DNNs and their relations to shallow PWLNN representations.

Activation	Expression $\sigma(\cdot)$	Description	Related Representations
ReLU ⁶	$\max\{x, 0\}$	retaining the positive part	CPLR, HH, SBF
Leaky ReLU ⁷⁶	$\max\{x, 0\} - \lambda \max\{-x, 0\}$	λ as a constant nonzero slope	CPLR, HH, SBF
Parametric ReLU ⁷⁷	$\max\{x_c, 0\} - \lambda_c \max\{-x_c, 0\}$	λ_c as trainable slopes in different channels x_c	CPLR, HH, SBF
Randomized ReLU ⁷⁸	$\max\{x_c^{(n)}, 0\} - \lambda_c^{(n)} \max\{-x_c^{(n)}, 0\}$	$\lambda^{(n)}$ as randomized slopes of sample $x^{(n)}$	CPLR, HH, SBF
Biased ReLU ⁷⁹	$\max\{0, x_i - b_{i1}\}, \dots, \max\{0, x_i - b_{iq_i}\}$	several biases for each input variable x_i	AHH, HL-CPLR
Concatenated ReLU ⁸⁰	$(\max\{x, 0\}, \max\{-x, 0\})$	concatenating both positive and negative parts	CPLR, HH, SBF
S-shaped ReLU ⁸¹	$\alpha_0 x + \beta_0 + \alpha_1 x - t^l + \alpha_2 x - t^r $	3 linear intervals with break points t^l and t^r	CPLR, HH, SBF
Flexible ReLU ⁸²	$\max\{x + a, 0\} + b$	a and b as trainable parameters	CPLR, HH, SBF
APL ⁸³	$\max\{x, 0\} + \sum_{i=1}^S a_i^+ \max\{0, -x + b_i^+\}$	a sum of S hinge-shaped functions	CPLR, HH, SBF
APRL ⁸⁴	$\lambda_R \max\{x, 0\} - \lambda_L \max\{-x, 0\}$	λ_R and λ_L as trainable slopes	CPLR, HH, SBF
Maxout ⁵¹	$\max_{i \in I} \{z_i\}$	maximal of multiple inputs	GHH

tree searching algorithm does not require computed gradients, and gives a novel interpretation of learning. In classic decision trees, the concept of PWL approximation can be applied to fit linear models on nodes⁸⁸. However, these local linear regressions bring substantially higher computational burden. A novel PWL decision tree as a flexible and efficient alternative has been previously constructed⁸⁹, which can be regarded as the extension of ReLU to the learning framework of decision trees.

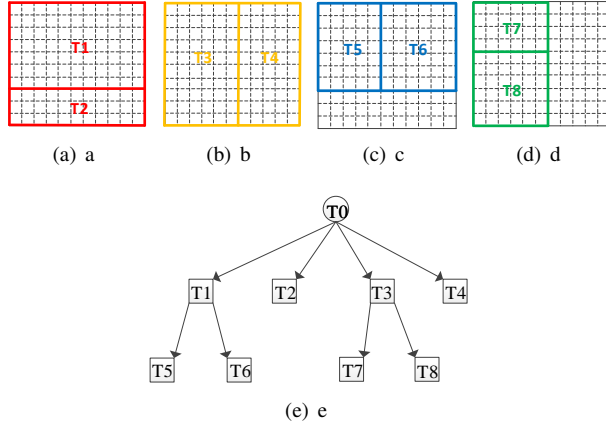


Figure 5. A simple illustration on the geometrical description and tree searching of learning an adaptive hinging hyperplane (AHH). a) Iteration 1 where $B_1(\mathbf{x}) = \max\{0, x_2 - 0.3\}$, $B_2(\mathbf{x}) = \max\{0, 0.3 - x_2\}$; b) Iteration 2 where $B_3(\mathbf{x}) = \max\{0, 0.6 - x_1\}$; c) Iteration 3 where $B_4(\mathbf{x}) = \max\{0, x_1 - 0.6\}$, $B_5(\mathbf{x}) = \min\{B_1, B_3\}$, $B_6(\mathbf{x}) = \min\{B_1, B_4\}$; d) Iteration 4 where $B_7(\mathbf{x}) = \min\{B_1, B_3\}$, and $B_8(\mathbf{x}) = \min\{B_2, B_3\}$; e) the resulting generic tree topology.

3.1.3 Structured Decision Algorithm

The basis functions in SBF are simplices and their shapes are controlled by $\gamma_{m,i}$ and $\zeta_{m,i}$ (Figure 6).

SBF takes the structures of basis functions as decision variables to design the learning algorithm rather than directly

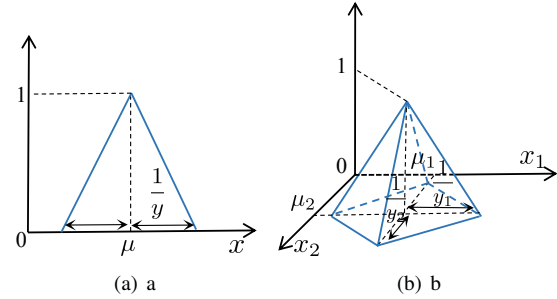


Figure 6. An illustration on the basis functions of simplex basis function (SBF) representation. a) One-dimensional SBF; b) Two-dimensional SBF.

taking the coefficient w_m as decision variables. In each iteration, the current highest peak point is selected as the center ζ_i . It then seeks the optimal structure of the simplices to minimize the overall approximation error¹⁷.

3.1.4 Other Algorithms

PWLNNs can be learned by interpolating the vertices over simplices and performing local linear approximation. Following this idea, HL-CPLR can be adopted. Without specifying domain configurations, linear approximation in sub-regions has been successfully applied in other learning algorithms for PWLNNs. For example, bi-level algorithms can be used, but they are only suitable in low dimensions with a few sampling data^{90,91}. For the Lattice, the structure could be identified by repeatedly updating the parameters by local fitting, where neither numerical simulations nor practical implementations were given²⁸.

3.2 Learning PWL-DNNs

In PWL-DNNs, the optimization problem is highly nonconvex and complicated, particularly for high-dimensional and large-scale data. Fortunately, the stacking network topology has enabled the optimization problem to be well resolved by successful applications of backpropagation and SGD.

3.2.1 Optimization of Network Parameters

With backpropagation, the chain rule can be applied to compute the gradients of learnable parameters, and then gradient-based algorithms can be applied. Specifically, denoting the weighted input of the k -th layer as \mathbf{z}_k and the output of the k -th layer as the activation σ_k for $k = 1, \dots, K$, see equation (22). The activation σ_k and the derivatives evaluated at \mathbf{z}_k in each layer are computed to be cached and then used to complete the backward propagation of gradients by the chain rule. In terms of matrix multiplication, the derivative of the loss with respect to the inputs is given by

$$\frac{\partial \mathcal{L}(g(\mathbf{x}|\mathcal{N}))}{\partial \sigma_K} \cdot \frac{\partial \sigma_K}{\partial \mathbf{z}_K} \cdot \frac{\partial \mathbf{z}_K}{\partial \sigma_{K-1}} \cdots \frac{\partial \sigma_1}{\partial \mathbf{z}_1} \cdot \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}}, \quad (24)$$

where $\mathcal{L}(\cdot)$ is a general loss and $g(\mathbf{x}|\mathcal{N})$ is the PWL-DNN.

In such learning schemes to compute the gradients in PWL-DNNs, powerful graphical or tensor processing unit hardware can be equipped to greatly accelerate the computation in the learning process. The massive amounts of data and considerably deep architectures make the optimization problems much more complicated, hence the computational efficiency of gradient descent algorithms is limited. For this reason, the stochastic version, namely SGD⁹², is the mainstream algorithm in deep learning, in which the batch-wise optimization accelerates the computation and improves the generalization ability. It also helps escape from the saddle points efficiently⁹³. After the success of SGD, numerous variants were developed with more sophisticated techniques to further enhance the performance in deep learning, such as adaptive learning rates⁹⁴, gradients with momentum⁹⁵, preconditioned gradients⁹⁶, and second-order methods⁹⁷.

In DNNs, the gradient vanishing problem is devastating, when the computation of gradients is propagated over layers. Fortunately, this problem is greatly alleviated by using ReLU⁷¹, allowing PWL-DNNs to be designed fairly deep (more than hundreds of layers)³². Beside the gradient-based algorithms for parameters, a robust initialization method is constructed by giving specified considerations to PWL nonlinearity, which enables to train extremely deep networks from scratch⁷⁷. Other useful techniques for generic DNNs can be utilized to improve the learning performance of PWL-DNNs, such as dropout⁹⁸, batch normalization⁹⁹, data augmentation¹⁰⁰, and pre-training¹⁰¹.

Concerning the PWL nonlinearity in deep learning, specialized algorithms have also been studied by utilizing the different local linear expressions of activated neurons to design novel algorithms. An adaptation of path-SGD is proposed to learn plain recurrent neural networks with ReLU, capturing long-term dependency structure and significantly improving the learnability¹⁰². By defining a transformation space of the positive scaling operators from ReLU, a modified SGD can be constructed in such a space, outperforming the conventional SGD¹⁰³. By assuming linearly separable data and utilizing local linearity of ReLU, a novel SGD is also constructed¹⁰⁴. Further, by partitioning node inputs of PWL-DNNs, the con-

vex hull over the partitions can be obtained, and thus mixed-integer and even convex optimization can be performed^{105, 106}. However, these algorithms currently require stringent assumptions and simple network structures with only a few layers, and yet are inapplicable to the learning of generic PWL-DNNs. Nevertheless, they innovate to exploit the benefits and utilize the special characteristics of PWL nonlinearity, promoting more in-depth understandings in learning deep architectures.

3.2.2 Optimization of Network Structures

Currently, the regular learning of PWL-DNNs is inherited from generic DNNs, where network structures are predefined through trial and error procedures, based on the researchers' prior knowledge and experience. The learnable parameters of interconnection weights and bias terms are then optimized with the aforementioned algorithms. In contrast, the learning algorithms in shallow PWLNNs involve simultaneous determination of both network structures and parameters, and they are model dependant by incorporating specific characteristics in each representation.

The greater flexibility of DNNs results in substantial redundancy in the network structures, leading to increasing efforts to optimize network structures in recent years. There have been numerous studies on compressing DNNs while guaranteeing similar or even better accuracy, such as sparsity regularization¹⁰⁷, neuron connections pruning¹⁰⁸ and low rank approximation¹⁰⁹. Recently, the lottery ticket hypothesis shows that randomly-initialized DNNs contain sub-networks (winning tickets) which are capable of reaching similar accuracy when being trained in isolation¹¹⁰. Rather than extracting a more compressed model from an existing DNN, Neural Architecture Search (NAS)¹¹¹ is proposed to gradually generate DNNs by searching the optimal building block candidates from a predefined search space via well-designed evaluation strategies. Although these algorithms are not limited to PWL-DNNs, they commonly adopt ReLU to introduce nonlinearity.

In fact, algorithms for the optimized network structures can also be found in the learning of different shallow PWLNNs, such as the l_1 norm sparsity regularization¹¹² and the backward pruning¹⁶. NAS is an incremental design, but seeks for both width and depth in network architectures, instead of merely growing the width in shallow architectures. In NAS, the candidate building blocks are network cells, which are commonly chosen as PWL sub-networks and are much more complicated, compared to those simple formulations of PWL basis functions. Though shallow PWLNNs are limited in empirical performances and problem scales, their learnings are also strongly related to current deep learning.

The success of effective learning techniques in deep learning now fundamentally realizes the pervasive applications of PWLNNs. In turn, the PWL-DNN itself also boosts thriving developments to generic DNNs, leading to a win-win situation between deep learning and PWLNN methods.

Table 2. Comparison on the universal approximation and representation abilities of different PWLNN representations.

Representation Model	Approximation	Representation
Conventional Representation ³	✓	✓
CPLR ⁵	✓	only for $\mathbf{x} \in \mathbb{R}$
G-CPLR ³⁴	✓	✓
HL-CPLR ⁶⁰	✓	✗
HH ⁹	✓	only for $\mathbf{x} \in \mathbb{R}$
GHH ¹⁴	✓	✓
AHH ¹⁶	✓	✗
SBF ¹⁷	✓	only for $\mathbf{x} \in \mathbb{R}$
Lattice ¹¹	✓	✓

3.3 Analysis on Shallow PWLNNs

In shallow PWLNNs, the ability to approximate arbitrary continuous functions (approximation ability) and the ability to represent arbitrary PWL functions (representation ability) have been the main focus in theoretical analysis.

3.3.1 Universal Approximation and Representation Abilities

The approximation and representation abilities are two different but important aspects in reflecting the properties of a class of PWLNNs in the form of a specific representation model. Table 2 summarizes these two properties.

Compared to the universal approximation, representation ability is more difficult to attain for a shallow PWLNN representation. It has been proven that the aforementioned representation models of shallow PWLNNs have universal approximation ability for continuous functions, but not necessarily have universal representation ability for continuous PWL functions, such as the counter example in equation (10). More specifically, given a continuous function and a specific PWLNN representation model having universal approximation ability, there always exist proper PWLNNs that can approximate this continuous function to arbitrary accuracy, while given a certain PWLNN as an approximator for this continuous function, such PWLNN might not be expressed in the form of this specific representation model. Hence, it is of great significance to select a representation model when approximating a nonlinear system with sampled data, since different selected representation models can lead to different PWLNNs with varied properties and difficulties in implementing the approximation¹⁹.

3.3.2 Model Properties and Their Connections

As the pioneering compact representation, the existence conditions of CPLR are as follows. A PWL function $f(\mathbf{x})$ has a CPLR representation of equation (7) if and only if it satisfies the consistent variation property⁴ [G]. A PWL function $f(\mathbf{x})$ possesses the consistent variation property if and only if $f(\mathbf{x})$ is partitioned by a finite set of hyperplanes $\mathcal{H}_k = \{\mathbf{x} \in \mathbb{R}^n : \pi_k(\mathbf{x}) := \alpha_k^T \mathbf{x} - \beta_k = 0\}$, for $k = 1, \dots, h; \forall k = 1, \dots, h$ partitioning hyperplane $\mathcal{H}_k = \{\mathbf{x} \in \mathbb{R}^n : \pi_k(\mathbf{x}) := \alpha_k^T \mathbf{x} - \beta_k = 0\}$, the dyadic product of any two local linear functions intersecting at the common boundary \mathcal{H}_k , meaning that $\Delta \mathbf{J}^{(i,j)} \doteq$

$\mathbf{J}_i - \mathbf{J}_j = c^{(i,j)} \boldsymbol{\alpha}$ and $\Delta b^{(i,j)} \doteq b_i - b_j = c^{(i,j)} \beta$, $c^{(i,j)} \in \mathbb{R}$ remains consistent. For every pair of neighboring sub-regions separated by a common boundary, the intersection between such two local linear functions must be a subset of an $(n-1)$ -dimensional hyperplane and cannot be covered by any hyperplane of lower dimensions.

When constructing PWLNNs using the Lattice representation, some interesting and fundamental properties are obtained. Firstly, the Lattice representation (any PWL function) can be transformed into the difference of two convex PWL functions¹⁹. For example, for any positive integers n and d , nonempty index sets $S_i \subseteq \mathbf{Z}(M), i \in \mathbf{Z}(M)$ and real vectors $\boldsymbol{\theta}(j) \in \mathbb{R}^{n+1}, j \in \mathbf{Z}(d)$, there always exist $n+1$ -dimensional real vectors $\boldsymbol{\theta}_+(k), k \in \mathbf{Z}(m_+)$ and $\boldsymbol{\theta}_-(k), k \in \mathbf{Z}(m_-)$ such that

$$\begin{aligned} & \min_{i \in \mathbf{Z}(M)} \max_{j \in S_i} \{l(\mathbf{x}|\boldsymbol{\theta}(j))\} \\ & = \max_{k \in \mathbf{Z}(m_+)} l(\mathbf{x}|\boldsymbol{\theta}_+(k)) - \max_{k \in \mathbf{Z}(m_-)} l(\mathbf{x}|\boldsymbol{\theta}_-(k)), \quad \forall \mathbf{x} \in \mathbb{R}^n, \end{aligned} \quad (25)$$

with $l(\mathbf{x}|\boldsymbol{\theta}) = \mathbf{J}^T \mathbf{x} + b$.

The above property indicates that any PWL function can be expressed as a difference of two convex PWL functions, each of which is formulated as the maximum of multiple linear functions¹⁹. On this basis, the proposal of GHH is promoted¹⁴, where the basis functions resemble the convex items (equation (25)). Moreover, in the analysis of GHH, the upper bound on the nesting number of G-CPLR for describing all PWL functions is significantly tightened.

The presented analysis reveals how these shallow PWLNN representations develop, how they are correlated and different to each other, and what important properties they possess, in terms of theoretical significance.

3.4 Analysis on PWL-DNNs

In PWL-DNNs, the approximation ability is reconsidered under the framework of deep learning and novel bounds are derived concerning both network width and depth. Rather than the representation ability for shallow PWLNNs, theoretical analysis for PWL-DNNs are mainly cast towards the learning process.

3.4.1 Approximation Ability Concerning Width and Depth

In early studies on approximation, neural networks containing one hidden layer with sigmoidal activations were favored¹¹³; this universal approximation ability also holds for PWLNNs^{9,74}. Universal approximation ability has been proven for two-hidden-layered neural networks¹¹⁴, and an estimation of hidden neurons was determined, rather than assuming an unbounded number of neurons^{113,115}. With success of ReLU, such approximation ability was further elucidated and specified for PWL-DNNs¹¹⁶. The fully connected PWL-DNNs with ReLU are universal approximators¹¹⁷, and one variant of ResNets has also been proven as a universal approximator when the depth of the network approaches infinity¹¹⁸.

It is also of great importance to investigate the relationship between the approximation behavior and their architectures, including width and depth. In early studies, the relationship between the approximation error and the number of neurons in the single hidden layer with sigmoid activations was investigated¹¹⁹. Specific Convolutional DNNs (namely ConvNet, ConvNet with ReLU and max pooling) are universal approximators, where the lower bound of the number of hidden channels is also provided¹²⁰. Further, by jointly considering depth and width of layers, the bounds on the number of layers and the number of neurons in hidden layers can be derived simultaneously³⁵. From the perspective of input space partitions, fully connected PWL-DNNs with ReLU can be equivalently transformed into two-layer fully connected DNNs, and the bounds on the width of the network to ensure the universal approximation ability can be determined¹²¹. These aforementioned works utilize the properties of PWL nonlinearity which allows outcomes such as partitioned linear sub-regions, activated linear outputs over layers, PWL finite element spaces, to name a few.

Current analysis on the approximation analysis in deep learning sheds more light on PWL-DNNs mostly with ReLU. They are heavily used in numerical settings, and more amenable to analysis, providing novel perspectives to broaden the existing theoretical understandings on the empirical learning process of DNNs. More rigorous discussions on the approximation analysis for PWLNNs are presented elsewhere¹²².

3.4.2 Expressive Capacity Given with Samples

The expressive capacity of DNNs commonly refers to the design of networks being able to realize arbitrary functions over a finite subset of the input space. In early studies, the expressive capacity of neural networks with one hidden layer and the sign activation were investigated¹²³, where the focus was on the injective functions¹²⁴. A two-hidden-layer neural network with sigmoid was proven to be capable of learning any N samples with arbitrary accuracy only if there are at least $2\sqrt{(p+2)N}$ neurons in the hidden layers, where p is the output dimension¹²⁵. For PWL-DNNs, relevant analysis is also conducted. In PWL-DNNs with ReLU, the expressive capacity can be guaranteed with the depth k , width $\mathcal{O}(N/k)$, and $\mathcal{O}(N+n)$ weights on a sample of size N in n dimensions¹²⁶. For arbitrary N samples, any pair of which has a specified minimum Euclidean distance, there exists a multi-layer ResNet that can express the samples accurately with only convolutional layers and ReLU activation functions¹²⁷. The expressive capability of the multi-layer ConvNet¹²⁸ and fully connected PWL-DNNs with ReLU¹²⁹ is also known. Though the above informative analysis helps to understand PWL-DNNs and even generic DNNs, the existing results still require tremendous neurons, particularly when large amounts of data are involved.

3.4.3 Analysis Specified with Localized Linearity over Domain Configurations

In PWL-DNNs, the unique property of localized nonlinearity makes it possible to quantitatively analyze the capacity of DNNs regarding domain configurations; a larger number of linear sub-regions indicates greater flexibility. The network expressive capacity can be evaluated by estimating the maximum number of linear sub-regions of fully connected PWL-DNNs with ReLU¹³⁰. The basic idea is to use the Zaslavsky's Theorem of hyperplanes arrangement¹³¹ [G], which estimates the maximal number of regions in \mathbb{R}^n with an arrangement of m hyperplanes. In PWL-DNNs, where the retained positive neuron output of the linear function in ReLU corresponds to these hyperplanes, this result can then be applied by recursively reusing the PWL fractures from the previous layers. By identifying distinct linear sub-regions, bounds for the maximal number of sub-regions can be derived. Various theoretical results and empirical studies on estimating the number of linear sub-regions can be obtained. For example, specific neuron allocation in each layer with the lower bound³⁵ or upper bound¹³² can be derived analogously. These bounds can be further improved with mild assumptions, and give more accurate evaluation in exploring the capacity of DNNs in this regard^{133,134}. In deep architectures, fully connected PWL-DNNs can be extended to ConvNet¹³⁵, where ConvNet with ReLU brings more linear sub-regions than that of fully connected PWL-DNNs with asymptotically the same number of parameters, input dimension, and number of layers¹³⁵. Such analyses consider the particular measurement of counting linear sub-regions in PWL-DNNs, and cannot be performed in DNNs with other types of nonlinearity, demonstrating the unique merit of PWL nonlinearity and providing a novel quantitative measurement of showing the power of going deep.

Although DNNs exhibit great flexibility, they have been found to be vulnerable to adversarial samples, meaning that human-imperceptible perturbation imposed on an image can fool the DNN to make a wrong prediction¹³⁶. Recent work on robustness certification focuses on whether the prediction of any input \mathbf{x} is verifiably constant within a neighboring set of \mathbf{x} ¹³⁷⁻¹³⁹. Most robustness certification is conducted with ReLU and heavily relies on the PWL nonlinearity for analysis and algorithm designs, such as the mix-integer linear programming^{140,141} and the convex outer approximation technique¹⁴², where the local information of linear boundaries and vertices is utilized. Hence, PWL nonlinearity also plays an essential role to advocate the robustness analysis in deep learning.

4 Applications

In this section, representative applications in different fields are introduced to exemplify the practical values of PWLNNs.

4.1 Circuits analysis

The pioneering work of shallow PWLNNs, particularly the canonical family, stems from the field of circuit analysis. It was first proposed to use vertices of simplices for the inter-

polation analysis of nonlinear resistive networks in 1956¹⁴³. The PWL technique has since attracted more attention and found successful applications in nonlinear circuit analysis involving uncoupled and monotonically increasing PWL resistors¹⁴⁴. This work has further been extended with rigorous improvements in circuit analysis, including more general solutions of equilibrium points^{145–150}, variously characterized electronic devices^{86,151–153} and circuits with complex dynamics^{154–158}. In recent years, efforts have been made to apply PWL techniques to the circuit systems containing memristors. PWL window functions are very flexible and have been applied to model different types of ideal memristors^{159–161}. The input-output data are also adopted to construct complex memristor systems from the perspective of approximation, in which PWLNNs in the form of SBF have been successfully applied with promising performance¹⁶². Many circuit components, such as resistors and memristors, are PWL-characterized, meaning that PWLNN relating methods can be naturally applied and good performance can commonly be expected.

4.2 Control

One of the most successful achievements of PWLNNs is their application in control systems, particularly in dynamic system identification and MPC. General PWLNN models naturally and widely exist in control systems, where hybrid phenomena can be handled^{163–166}. The systems here are not necessarily continuous, albeit they are PWL-characterized, the discontinuous cases are not discussed in this Primer.

System identification consists of building a proper mathematical model to describe the coherent relationship based on the given input-output data from a dynamic control system. Note that the basic idea in system identification is similar to that of general approximation problems and supervised learning. Here, we mainly discuss the PWLNNs applied in typical problems concerning control community, where the dynamic system is in relatively lower dimensions and the dynamics of control systems are addressed. PWLNNs have shown great potentials in various dynamic systems^{27–29}. In fact, many learning algorithms specified for shallow PWLNNs initially originated from resolving the system identification problems in control^{16,17,31,167,168}. Recently, a novel PWLNN for dynamic system identification was developed to provide an interpretable predictor facilitating variable selection and analysis⁸⁵, and has been applied to traffic flow prediction¹⁶⁹.

MPC has long been addressed as an important topic for complex constrained multi-variable control problems, particularly in industrial processes¹⁷⁰. The controller design in MPC is an open-loop optimization problem. The intensive online computation of the repetitive solutions to the optimization problem is a major obstacle hindering its wider use. Explicit solutions to linear MPC have been proposed, in which the controller is formulated as a PWL function of state variables^{171–173}. It has become more prevalent to use PWL-characterized predictive models in MPC, where different PWLNNs have been applied.

In one example, the power of PWLNNs in MPC was preliminarily verified by using hinging hyperplanes, but this method neither fully considered the superiority of PWLNN nor exerted a good combination with MPC¹⁷⁴. The Lattice representation was then utilized to achieve analytical expression of the explicit MPC solution¹⁷⁵. AHH was also successfully applied to determine the necessary and sufficient conditions for local optimality¹⁶. Further, the minimal conjunctive normal expression based on the Lattice was achieved for MPC with the smallest number of parameters²⁵. It has also been shown that using convex projections of the Lattice representation can be another potential technique for solving explicit MPC¹⁷⁶. Compared to other methods, PWLNNs have relatively simpler identification process and also show advantageous accuracy with efficiency. It inspires us that a good utilization of specific geometrical properties of PWLNNs and a proper combination with practical settings fundamentally help approach more extensive applications.

4.3 Image processing

Nowadays, PWL-DNNs can be designed over hundreds of layers to extract informative features for the learning of complicated tasks³³. PWL-DNNs have become one of the most popular choices in deep learning, where the state-of-the-art performance of many popular image datasets, such as ImageNet, has been constantly refreshed along with the proposal of various PWL activations or related techniques^{51,76}. In addition to image processing, acoustic and video processing can be improved based on PWL-DNNs^{177,178}.

In 2012, a PWL-DNN with ReLU was applied to greatly improve the classification accuracy on approximately 1.2 million images into 1000 classes³². By introducing ReLU to DNNs, the gradient vanishing could be greatly relieved with boosted performances, resulting in faster learning compared to Tanh units and others³². In fact, faster learning has a great influence on the performance of large models trained on large-scale data. PWL-DNNs have long been dominating the image processing tasks and constantly improving accuracy in various benchmarks^{77,179}. Practical real-world applications have received great benefits from the commercial aspects of hardware and algorithmic implementations^{75,180}. PWL-DNNs have made and will continue making remarkable contributions to image processing and far beyond.

5 Reproducibility and data deposition

To apply PWLNNs in data science, reproducibility and data deposition is an important element. For a specific task with given data, a general workflow to sequentially apply a PWLNN consists of selecting a PWLNN representation model, feeding data into the PWLNN, conducting the learning, evaluating the performance and tuning until the desired conditions are satisfied (Figure 1). In such a workflow, there are many factors affecting the reproducibility, which can be categorized into data influence and algorithmic influence.

5.1 Data influence

When a new algorithm or model is proposed and needs to be verified, the baselines of data sources are required. Thus, various carefully designed benchmark datasets are collected from natural environments and have been widely used for applying PWLNNs (including other methods for data) to validate their effectiveness with fair comparisons. When using benchmark datasets, reproducibility can be guaranteed. In practical scenarios, particularly involving industrial processes, data can be corrupted with noise, due to environmental influences or measurement errors. Thus, sometimes in the evaluation, the performances against noises need to be considered. In such cases, to ensure reproducibility, manually-corrupted noises by computer programming can be made identical by fixing the computer seed for generating noise in the programming of each run. In this way, benchmark datasets together with identical noises can be evaluated by computer runs. In deep learning, there are numerous benchmark datasets, such as UCI repository¹⁸¹, MNIST¹⁸², SVHN¹⁸³, NORB¹⁸⁴, CIFAR 10/100¹⁸⁵, COCO¹⁸⁶, and ImageNet¹⁸⁷, Visual Genome¹⁸⁸ meaning that solid experiments of empirical evaluations can be conducted with guaranteed reproducibility.

Algorithmic influence

Learning algorithms are also significant for reproducibility since different sets of parameters can result in a completely different performance even with the same PWLNN structure and identical dataset. Random initialization can lead to different results in learning algorithms, as parameters get iteratively updated to different values. Therefore, the same random seed should be fixed for the initialization of learning algorithms. In particular, the learning algorithms for PWL-DNNs are based on gradients, the computation of which can differ in concrete implementations, because the gradients are usually computed numerically due to the nondifferentiable PWL nonlinearity. Various computational platforms are constructed, where PWL-DNNs can be easily applied and well learned. Each platform has its own standards; popular platforms include Tensorflow¹⁸⁹, PyTorch³⁶, Keras¹⁹⁰, Caffe¹⁹¹, MXNet¹⁹², Theano¹⁹³. Once a certain platform is selected, reproducibility can be guaranteed. Data decomposition can also be implemented identically if all randomness has been eliminated by the previously mentioned strategies.

6 Limitations and optimizations

Despite the developments of PWLNNs achieved so far, there are still many challenges worthy to be addressed. In shallow PWLNNs, there are different ways of introducing PWL nonlinearity. For example, each PWLNN representation based on basis functions has distinctively different motivations and explanations for the varied PWL feature mappings. In contrast, the PWL nonlinearity in PWL-DNNs is achieved by directly adopting some simple PWL functions as activations. In existing PWL-DNNs, the basic building blocks of the network and connections between neurons, both within and across layers,

are inherited from generic DNNs. They lack specifications by combining PWL nonlinearity on constructing novel deep architectures \mathcal{N} . The potential of extending PWLNNs in the form of AHH towards novel deep network architectures has been explored¹⁹⁴, but it still remains difficult to tackle complex tasks involving really deep architectures.

To apply PWLNNs in data science, various learning algorithms have been proposed. It should be noted that although shallow PWLNNs are only applicable to a limited range of problems in low dimensions and with small scales, each PWLNN representation model has its own learning algorithm, which is constructed by fully considering the specific characteristics of the model. When it comes to PWL-DNNs, we admit that the combination of SGD and backpropagation in deep learning has truly realized the powerful flexibility of PWL-DNNs and promoted the applications, but the regular learning algorithms for PWL-DNNs hold no difference from other generic DNNs. In fact, when it involves an optimization problem with PWL nonlinearity, local information of vertices and linearity have not been utilized, and yet it shows to be quite useful and promising in solving the related optimization problems of shallow PWLNNs for attaining higher accuracy, efficient computation, and explicitness of learning process. We can rethink developing novel learning algorithms for PWL-DNNs with specifications on different PWL activation functions or specific network architectures, in which full information from PWL nonlinearity should be utilized and more potentials are promising to be further explored.

In the existing literature, theoretical analysis concerning shallow PWLNN representations has been discussed vigorously, such as representation ability, existence conditions, and domain configurations. As pointed out in this Primer, strong relations exist between the shallow and deep PWLNNs. However, only a few of the theoretical conclusions for the existing PWLNN representations have been used in deep learning. For example, the universal representation ability of GHH in analyzing shallow PWLNNs has been employed to understand PWL-DNNs with ReLU³⁵. PWL-DNNs shall be further investigated by recalling vigorous theories in the previously existing PWLNNs to facilitate further understanding of PWL-DNNs and even generic DNNs.

7 Outlook

An implicit and practical requirement for conventional PWL functions is to be compact and interpretable with regards to domain partitions and locally linear expressions, since complex partitions and expressions might result in PWL functions that are hard to be understood with unpredictable behaviors. As a result, shallow PWLNNs usually assume locally-dominant features among sub-regions and aim to achieve sufficiently sparse model structures. In contrast, PWL-DNNs abandon such an assumption and directly adopt simple PWL mapping functions to connect neurons in a certain way. This results in much more complicated domain partitions and locally linear expressions. Although numerical results demonstrate the

superior performance of PWL-DNNs, there are still many open problems in this field. In order to understand the fundamentals of DNNs and further improve practical applications, certain questions need to be answered. For example, given the data, is there a PWLNN that simultaneously pertains simple partitions and/or locally-dominant features with considerably good performances? Can we find such a PWLNN by explicitly seeking a shallow PWLNN or implicitly regularizing the learning of a PWL-DNN? What are the differences and relations between PWLNNs and other kinds of NNs that address locally-dominant features¹⁹⁵?

Despite the superior performance of PWL-DNNs, the shallow architectures discussed in this Primer show their own merits, including simpler structures, better interpretability, and the alleviated overfitting, particularly in lower dimensions with smaller-scale problems. Novel formulations of shallow-architecture PWLNNs are still worth being explored. For example, inserting more linear functions into the hinges for flexibility using GHH and utilizing max-min composites of univariate hinges for interpretability with AHH. The Lattice representation has not yet been extended to deep architectures, and can be a promising alternative to develop novel network architectures in deep learning. The Lattice representation can be transformed with strong relations to GHH, which has been extended to PWL-DNNs with Maxout. Though the boolean theory was discussed to formulate a multi-level Lattice network¹⁹⁶, the transformation is neither unique nor irredundant, and its composite of PWL mappings lacks flexibility, leaving potential for further improvement, such as large-scale explicit MPC and its efficient hardware implementation.

PWL nonlinearity has significantly contributed to the design of deeper architectures. Although there is an emerging line of empirical and theoretical works on PWL-DNNs, current theoretical analysis of PWL-DNNs is still far from sufficient, particularly for different variants of PWL mapping functions, compared to the existing theoretical understanding of their shallow counterparts, where specific analysis is cast for each PWLNN representation. In fact, there are many informative conclusions in shallow PWLNN representations that are very helpful for deep learning^{35,51}. More rigorous theoretical analysis can further benefit the understanding of PWL-DNNs and generic DNNs. Novel network architectures in deep learning are also promising to be inspired, when rethinking the existing systematical theories and experiences for general PWLNNs.

The learning of shallow PWLNNs is specified in each representation, while PWL-DNNs directly inherit the regular learning from generic DNNs, where PWL-specified techniques are missing. Thus, when developing novel PWL-DNNs, learning algorithms for PWL-DNNs should be taken into account by fully utilizing PWL characteristics specified on different types of PWL mapping functions, so that novel architectures and effective learning algorithms can be mutually developed and boosted. In the learning of DNNs, another challenge is to study how the parameter learning can converge to a local

or global optimum in such highly nonconvex optimization problems. Optimality conditions have been proven for linear DNNs^{197,198}, and nonlinear DNNs mostly with differentiable activations^{199,200}. Current analytical results rely on simple DNNs restricted to multiple assumptions. Considering the superior performances of nondifferentiable PWL-DNNs, related analysis of their optimization is worthy of further attention. It is necessary to analyze these optimality conditions specified on different types of (PWL) activations and more complex network structures for specific objectives.

In this Primer, we comprehensively introduced PWLNNs and their developments since 1970s, from tracing to the pioneering PWLNNs in the form of canonical representations, subsequent studies on shallow architectures, and the recent developments in deep ones. In the upcoming years, PWLNNs are sure to be of great significance, and vigorous developments should be expected, particularly in such an era with massive information in-and-out, which keeps benefiting our society.

8 Glossary

An induced conclusion by the Stone-Weierstrass approximation theorem: any continuous function can be approximated by a PWL function to arbitrary accuracy.

PWL function: it is a function that appears to be linear in sub-regions of the domain but is by essence nonlinear in the whole domain.

Canonical piecewise Linear Representation: it is the pioneering compact expression by which a PWL function is constructed through a linear combination of multiple absolute-value basis functions.

Rectified linear units: it is one of the most popular activation functions in neural networks and is defined as the positive part of its arguments by $\max\{0, x\}$.

Hinging hyperplanes: a hinge function consists of two hyperplanes, namely hinging hyperplanes, continuously joining at the so-called hinge, and has greatly contributed to construct flexible representation models for continuous PWL functions.

Backpropagation strategy: it is widely used to train feed-forward neural networks and works by computing the gradients of weights of each layer in the network and iterating backward layerwisely for efficient calculation.

Stochastic gradient descent: it is an iterative optimization algorithm, where the actual gradient is approximated or estimated commonly by a randomly selected subset of data.

PWL memristors: other than the resistor, the inductor, and the capacitor, it is considered as the fourth fundamental two-terminal circuit element including a memory of past voltages or currents; those memristors pertain PWL-characterized dynamics are called PWL memristors.

Gradient vanishing problem: in the iterative updates of training DNNs with gradient-based algorithms, the multiplying of small values of gradients by backpropagation can lead to a very small value (approaching zero) in computing the

gradients of early layers, which makes the network hard to proceed the training.

Least squares method: it is an approach to approximate the solutions of an unknown system given with a set of input-output data points by minimizing the sum of the squares of the residuals between the observed output data and network's output.

Gauss-Newton algorithm: it is a modified Newton's method, which computes the second-order derivatives, to minimize a sum of squared loss in solving non-linear least squares problems.

Multivariate adaptive regression splines: it is a flexible regression model, consisting of weighted basis functions, which are expressed in terms of the product of truncated power splines $[\pm(x_i - \beta)]_+^q$, and its training procedures can be interpreted as a generalized tree searching based on recursive domain partitions.

Consistent variation property: given a continuous PWL function, it is the necessary and sufficient condition on whether such a function can be expressed by a CPLR model, where the properties of domain partitions and intersections between partitioned sub-regions are discussed; its detailed descriptions are given in the subsequent context.

Zaslavsky's Theorem of hyperplanes arrangement: the maximal number of regions in \mathbb{R}^d with an arrangement of m hyperplanes is estimated by $\sum_{j=0}^n \binom{m}{j}$.

9 Acknowledgements

This work is jointly supported by ERC Advanced Grant E-DUALITY (787960), KU Leuven Grant CoE PFV/10/002, and Grant FWO G0A4917N, EU H2020 ICT-48 Network TAILOR (Foundations of Trustworthy AI - Integrating Reasoning, Learning and Optimization), Leuven.AI Institute, National Key Research and Development Program under Grant 2021YFB2501200, and Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

References

1. Leenaerts, D. & Van Bokhoven, W. M. *Piecewise linear modeling and analysis* (Springer Science & Business Media, 2013).
2. Folland, G. B. *Real Analysis: Modern Techniques and Their Applications* (Wiley Interscience, 1999).
3. Chien, M.-J. & Kuh, E. Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision. *IEEE Transactions on Circuits Syst.* **24**, 305–317 (1977).
4. Chua, L. O. & Deng, A. Canonical piecewise-linear representation. *IEEE Transactions on Circuits Syst.* **35**, 101–111 (1988). **The systematical analysis on CPLR is given in the paper, including some crucial properties of PWLNNs.**
5. Chua, L. O. & Kang, S. Section-wise piecewise-linear functions: Canonical representation, properties, and applications. *Proc. IEEE* **65**, 915–929 (1977). **The pioneering compact expression for PWL functions is proposed and formally introduced in this paper for circuit systems, and then the analytical analysis for PWL functions since becomes viable.**
6. Nair, V. & Hinton, G. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on International Conference on Machine Learning*, 807–814 (2010). **PWL-DNNs start being prevalent and showing state-of-the-art performance since this paper, where the most popular ReLU is established.**
7. Kang, S. & Chua, L. O. A global representation of multidimensional piecewise-linear functions with linear partitions. *IEEE Transactions on Circuits Syst.* **25**, 938–940 (1978).
8. Lin, J. N. & Unbehauen, R. Canonical representation: from piecewise-linear function to piecewise-smooth functions. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **40**, 461–468 (1993).
9. Breiman, L. Hinging hyperplanes for regression, classification, and function approximation. *IEEE Transactions on Inf. Theory* **39**, 999–1013 (1993). **HH representation model and its hinge-finding learning algorithm are introduced in this paper. The connection with ReLU in PWL-DNNs can be referred.**
10. Lin, J. N. & Unbehauen, R. Explicit piecewise-linear models. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **41**, 931–933 (1995).
11. Trela, J. & Martínez, M. Region configurations for realizability of lattice piecewise-linear models. *Math. Comput. Model.* **30**, 17–27 (1999). **Formal proofs on the universal representation ability of the Lattice representation are given and different locally linear sub-region realizations are summarized.**
12. Julián, P. The complete canonical piecewise-linear representation: Functional form for minimal degenerate intersections. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **50**, 387–396 (2003).
13. Wen, C., Wang, S., Li, F. & Khan, M. J. A compact f-f model of high-dimensional piecewise-linear function over a degenerate intersection. *IEEE Transactions on Circuits Syst. I: Regul. Pap.* **52**, 815–821 (2005).
14. Wang, S. & Sun, X. Generalization of hinging hyperplanes. *IEEE Transactions on Inf. Theory* **51**, 4425–4431 (2005). **The idea of inserting multiple linear functions to the hinge is given in this paper, where formal proofs are given. The connection with Max-out in PWL-DNNs can be referred.**

15. Sun, X. & Wang, S. A special kind of neural networks: Continuous piecewise linear functions. *Lect. Notes Comput. Sci.* **3496**, 375–379 (2005).
16. Xu, J., Huang, X. & Wang, S. Adaptive hinging hyperplanes and its applications in dynamic system identification. *Automatica* **45**, 2325–2332 (2009).
17. Yu, J., Wang, S. & Li, L. Incremental design of simplex basis function model for dynamic system identification. *IEEE Transactions on Neural Networks Learn. Syst.* **29**, 4758–4768 (2017).
18. Chua, O., L. & Deng, A. C. Canonical piecewise-linear analysis - Part II: Tracing driving-point and transfer characteristics. *IEEE Transactions on Circuits Syst.* **32**, 417–444 (1985).
19. Wang, S. General constructive representations for continuous piecewise-linear functions. *IEEE Transactions on Circuits Syst. I: Regul. Pap.* **51**, 1889–1896 (2004). **A general constructive method for representing an arbitrary PWL function is considered, in which significant differences and connections between different representation models are vigorously discussed. Many theoretical analysis on PWL-DNNs adopts the Theorems and Lemmas proposed in this paper.**
20. Wang, S., Huang, X. & Yam, Y. A neural network of smooth hinge functions. *IEEE Transactions on Neural Networks* **21**, 1381–1395 (2010).
21. Xu, J., Huang, X. & Wang, S. Stability analysis of planar continuous piecewise linear systems. In *Proceedings of the American Control Conference*, 2505–2510 (2010).
22. Mu, X., Huang, X. & Wang, S. Dynamic behavior of piecewise-linear approximations. *J. Tsinghua Univ.* **51**, 879–883 (2011).
23. Huang, X., Xu, J. & Wang, S. Exact penalty and optimality condition for nonseparable continuous piecewise linear programming. *J. Optim. Theory Appl.* **155**, 145–164 (2012).
24. Xu, J., Boom, T., Schutter, B. & Wang, S. Irredundant lattice representations of continuous piecewise affine functions. *Automatica* **70**, 109–120 (2016).
25. Xu, J., Boom, T., Schutter, B. & Luo, X. Minimal conjunctive normal expression of continuous piecewise affine functions. *IEEE Transactions on Autom. Control.* **61**, 1340–1345 (2016).
26. Pucar, P. & Millnert, M. Smooth hinging hyperplanes - an alternative to neural nets. In *Proceedings of the 3rd European Control Conference*, 1173–1178 (1995).
27. Hush, D. & Horne, B. Efficient algorithms for function approximation with piecewise linear sigmoidal networks. *IEEE Transactions on Neural Networks* **9**, 1129–1141 (1998).
28. Wang, S. & Narendra, K. S. Nonlinear system identification with lattice piecewise-linear functions. In *Proceedings of the American Control Conference*, 388–393 (2002).
29. Wen, C., Wang, S., Jin, X. & Ma, X. Identification of dynamic systems using piecewise-affine basis function models. *Automatica* **43**, 1824–1831 (2007).
30. Wang, S., Huang, X. & Khan Junaid, K. M. Configuration of continuous piecewise-linear neural networks. *IEEE Transactions on Neural Networks* **19**, 1431–45 (2008).
31. Huang, X., Xu, J. & Wang, S. Identification algorithm for standard continuous piecewise linear neural network. In *Proceedings of the American Control Conference*, 4431–4936 (2010). **A gradient descent learning algorithm is proposed for PWLNNs, where domain partitions and parameter optimizations are both elucidated.**
32. Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1097–1105 (2012).
33. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778 (2016).
34. Huang, G., Liu, Z., van der Maaten, L. & Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2261–2269 (2017).
35. Arora, R., Basu, A., Mianjy, P. & Mukherjee, A. Understanding deep neural networks with rectified linear units. In *Proceedings of the International Conference on Learning Representations* (2018).
36. Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 8024–8035 (2019).
37. Julián, P. *A high level canonical piecewise linear representation: theory and applications*. Ph.D. thesis, Universidad Nacional del Sur (Argentina) (1999). **This dissertation gives a very good view on the PWL functions and their applications mainly in circuits systems developed before the 2000s.**
38. Ohnishi, M. & Inaba, N. A singular bifurcation into instant chaos in a piecewise-linear circuit. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **41**, 433–442 (1994).
39. Itoh, M. & Chua, L. O. Memristor oscillators. *Int. J. Bifurc. Chaos* **18**, 3183–3206 (2008).
40. Bradley, P. S., Mangasarian, O. L. & Street, W. N. Clustering via concave minimization. In *Advances in Neural Information Processing Systems*, 368–374 (1996).

41. Kim, D. & Pardalos, P. M. A dynamic domain contraction algorithm for nonconvex piecewise linear network flow problems. *J. Glob. Optim.* **17**, 225–234 (2000).
42. Balakrishnan, A. & Graves, S. C. A composite algorithm for a concave-cost network flow problem. *Networks* **19**, 175–202 (2010).
43. Liu, K., Xu, Z., Xi, X. & Wang, S. Sparse signal reconstruction via concave continuous piecewise linear programming. *Digit. Signal Process.* **54**, 12–26 (2016).
44. Liu, K., Xi, X., Xu, Z. & Wang, S. A piecewise linear programming algorithm for sparse signal reconstruction. *Tsinghua Sci. Technol.* **22**, 29–41 (2017).
45. Zhang, H. & Wang, S. Global optimization of separable objective functions on convex polyhedra via piecewise-linear approximation. *J. Comput. Appl. Math.* **197**, 212–217 (2006).
46. Zhang, H. & Wang, S. Linearly constrained global optimization via piecewise-linear approximation. *J. Comput. Appl. Math.* **214**, 111–120 (2008).
47. Guisewite, G. M. & Pardalos, P. M. Minimum concave-cost network flow problems: Applications, complexity, and algorithms. *Annals Oper. Res.* **25**, 75–99 (1991).
48. Burkard, R. E., Dollani, H. & Thach, P. T. Linear approximations in a dynamic programming approach for the uncapacitated single-source minimum concave cost network flow problem in acyclic networks. *J. Glob. Optim.* **19**, 121–139 (2001).
49. Xi, X., Huang, X., Suykens, J. A. K. & Wang, S. Coordinate descent algorithm for ramp loss linear programming support vector machines. *Neural Process. Lett.* **43**, 887–903 (2016).
50. Xu, Z., Liu, K., Xi, X. & Wang, S. Method of hill tunneling via simplex centroid for continuous piecewise linear programming. In *Proceedings of the IEEE Conference on Decision and Control*, 6609–6616 (2015).
51. Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A. & Bengio, Y. Maxout networks. In *Proceedings of the International Conference on Machine Learning*, 1319–1327 (2013). **A flexible PWL activation function is proposed for PWL-DNNs, and ReLU can be regarded as its special case, where analysis on the universal approximation ability and the relations to the shallow-architected PWLNNs are given.**
52. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. national academy sciences* **79**, 2554–2558 (1982).
53. Kahlert, C. & Chua, L. O. A generalized canonical piecewise-linear representation. *IEEE Transactions on Circuits Syst.* **37**, 373–383 (1990).
54. Lin, J., Xu, H.-Q. & Unbehauen, R. A generalization of canonical piecewise-linear functions. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **41**, 345–347 (1994).
55. Ernst, S. Hinging hyperplane trees for approximation and identification. In *Proceedings of the IEEE Conference on Decision and Control*, vol. 2, 1266–1271 (1998).
56. Pucar, P. & Sjöberg, J. On the hinge-finding algorithm for hinging hyperplanes. *IEEE Transactions on Inf. Theory* **44**, 3310–3319 (1998).
57. Ramirez, D. R., Camacho, E. F. & Arahal, M. R. Implementation of min-max MPC using hinging hyperplanes. application to a heat exchanger. *Control. Eng. Pract.* **12**, 1197–1205 (2004).
58. Huang, X., Matijaš, M. & Suykens, J. A. Hinging hyperplanes for time-series segmentation. *IEEE Transactions on Neural Networks Learn. Syst.* **24**, 1279–1291 (2013).
59. Huang, X., Xu, J. & Wang, S. Operation optimization for centrifugal chiller plants using continuous piecewise linear programming. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 1121–1126 (2010).
60. Julián, P., Desages, A. & Agamennoni, O. High-level canonical piecewise linear representation using a simplicial partition. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **46**, 463–480 (1999).
61. Padberg, M. Approximating separable nonlinear functions via mixed zero-one programs. *Oper. Res. Lett.* **27**, 1–5 (2000).
62. Croxton, K. L., Gendron, B. & Magnanti, T. L. A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems. *Manag. Sci.* **49**, 1268–1273 (2003).
63. Keha, A. B., de Farias, I. R. & Nemhauser, G. L. A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization. *Oper. Res.* **54**, 847–858 (2006).
64. Vielma, J. P., Ahmed, S. & Nemhauser, G. Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Oper. research* **58**, 303–315 (2010).
65. Wilkinson, R. A method of generating functions of several variables using analog diode logic. *IEEE Transactions on Electron. Comput.* **12**, 112–129 (1963).
66. Birkhoff & Garrett. Lattice theory. *Bull. Am. Math. Soc.* **64**, 50–57 (1958).
67. Streubel, T., Griewank, A., Radons, M. & Bernt, J.-U. Representation and analysis of piecewise linear functions in abs-normal form. In *Proceedings of IFIP Conference on System Modeling and Optimization*, 327–336 (2013).

68. Griewank, A. On stable piecewise linearization and generalized algorithmic differentiation. *Optim. Methods Softw.* **28**, 1139–1178 (2013).
69. Fiege, S., Walther, A. & Griewank, A. An algorithm for nonsmooth optimization by successive piecewise linearization. *Math. Program.* **177**, 343–370 (2019).
70. Griewank, A. & Walther, A. Polyhedral DC decomposition and DCA optimization of piecewise linear functions. *Algorithms* **13**, 166 (2020).
71. Glorot, X., Bordes, A. & Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 315–323 (2011).
72. McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin mathematical biophysics* **5**, 115–133 (1943).
73. Batruni, R. A multilayer neural network with piecewise-linear structure and back-propagation learning. *IEEE Transactions on Neural Networks* **2**, 395–403 (1991).
74. Lin, J. N. & Unbehauen, R. Canonical piecewise-linear networks. *IEEE Transactions on Neural Networks* **6**, 43–50 (1995). **The network topology for G-CPLR is depicted, and the idea of introducing general PWL activation functions for PWL-DNNs is also discussed in this paper, yet without numerical evaluations.**
75. Rawat, W. & Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Comput.* **29**, 2352–2449 (2017).
76. Maas, A., Hannun, A. Y. & Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference Machine Learning*, 1–8 (2013).
77. He, K., Zhang, X., Ren, S. & Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 1026–1034 (2015). **Modifications of optimization strategies on the PWL-DNNs and a novel PWL activation function are given in this paper, where PWL-DNNs can be delved into fairly deep.**
78. Xu, B., Wang, N., Chen, T. & Li, M. Empirical evaluation of rectified activations in convolutional network. Preprint at <https://arxiv.org/abs/1505.00853> (2015).
79. Liang, X. & Xu, J. Biased ReLU neural networks. *Neurocomputing* **423**, 71–79 (2021).
80. Shang, W., Sohn, K., Almeida, D. & Lee, H. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *Proceedings of the International Conference on Machine Learning*, vol. 48, 2217–2225 (2016).
81. Jin, X. *et al.* Deep learning with s-shaped rectified linear activation units. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1737–1743 (2016).
82. Qiu, S., Xu, X. & Cai, B. FReLU: Flexible rectified linear units for improving convolutional neural networks. In *Proceedings of the International Conference on Pattern Recognition*, 1223–1228 (2018).
83. Agostinelli, F., Hoffman, M. D., Sadowski, P. J. & Baldi, P. Learning activation functions to improve deep neural networks. In *Workshop Track Proceedings of the International Conference on Learning Representations* (2015).
84. Bodyanskiy, Y., Deineko, A., Pliss, I. & Slepanska, V. Formal neuron based on adaptive parametric rectified linear activation function and its learning. In *Proceedings of the International Workshop on Digital Content & Smart Multimedia*, vol. 2533, 14–22 (2019).
85. Xu, J. *et al.* Efficient hinging hyperplanes neural network and its application in nonlinear system identification. *Automatica* **116**, 108906 (2020).
86. Suykens, J. A., Huang, A. & Chua, L. O. A family of n-scroll attractors from a generalized chua’s circuit. *Arch. fur Elektronik und Ubertragungstechnik (International J. Electron. Commun.* **51**, 131–137 (1997).
87. Friedman, J. H. *et al.* Multivariate adaptive regression splines. *The Annals Stat.* **19**, 1–67 (1991).
88. Wang, Y. & Witten, I. H. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning* (1997).
89. Tao, Q. *et al.* Learning with continuous piecewise linear decision trees. *Expert. Syst. with Appl.* **168**, 114–214 (2020).
90. Ferrari-Trecate, G., Muselli, M., Liberati, D. & Morari, M. A clustering technique for the identification of piecewise affine systems. *Automatica* **39**, 205–217 (2003).
91. Nakada, H., Takaba, K. & Katayama, T. Identification of piecewise affine systems based on statistical clustering technique. *Automatica* **41**, 905–913 (2005).
92. Bottou, L. Stochastic gradient learning in neural networks. *Proc. Neuro-Nimes* **91**, 12 (1991).
93. Jin, C., Netrapalli, P., Ge, R., Kakade, S. M. & Jordan, M. I. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. *J. ACM* **68**, 1–29 (2021).
94. Duchi, J., Hazan, E. & Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **12**, 2121–2159 (2011).
95. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations* (2015).

96. Gupta, V., Koren, T. & Singer, Y. Shampoo: Preconditioned stochastic tensor optimization. In *Proceedings of the International Conference on Machine Learning*, 1842–1850 (2018).
97. Anil, R., Gupta, V., Koren, T., Regan, K. & Singer, Y. Scalable second order optimization for deep learning. Preprint at <https://arxiv.org/abs/2002.09018> (2020).
98. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**, 1929–1958 (2014).
99. Ioffe, S. & Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, 448–456 (2015).
100. Shorten, C. & Khoshgoftaar, T. M. A survey on image data augmentation for deep learning. *J. Big Data* **6**, 1–48 (2019).
101. Erhan, D., Courville, A., Bengio, Y. & Vincent, P. Why does unsupervised pre-training help deep learning? In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 201–208 (2010).
102. Neyshabur, B., Wu, Y., Salakhutdinov, R. & Srebro, N. Path-normalized optimization of recurrent neural networks with ReLU activations. In *Advances in Neural Information Processing Systems*, 3477–3485 (2016).
103. Meng, Q. *et al.* G-SGD: optimizing relu neural networks in its positively scale-invariant space. In *Proceedings of the International Conference on Learning Representations* (2019).
104. Wang, G., Giannakis, G. B. & Chen, J. Learning relu networks on linearly separable data: Algorithm, optimality, and generalization. *IEEE Transactions on Signal Process.* **67**, 2357–2370 (2019).
105. Tsay, C., Kronqvist, J., Thebelt, A. & Misener, R. Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. In *Advances in Neural Information Processing Systems*, vol. 34, 2993–3003 (2021).
106. Ergen, T. & Pilanci, M. Global optimality beyond two layers: Training deep relu networks via convex programs. In *International Conference on Machine Learning*, 2993–3003 (2021).
107. Wen, W., Wu, C., Wang, Y., Chen, Y. & Li, H. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, 2074–2082 (2016).
108. Han, S., Pool, J., Tran, J. & Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, 1135–1143 (2015).
109. Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y. & Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. 1269–1277 (2014).
110. Frankle, J. & Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations*, 6336–6347 (2019).
111. Zoph, B. & Le, Q. V. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations* (2017).
112. Tao, Q., Xu, J., Suykens, J. A. K. & Wang, S. Fast adaptive hinging hyperplanes. In *Proceedings of the IEEE Conference on Decision and Control*, 1482–1487 (2018).
113. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.* **2**, 303–314 (1989).
114. Kurková, V. Kolmogorov’s theorem and multilayer neural networks. *Neural networks* **5**, 501–506 (1992).
115. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **2**, 359–366 (1989).
116. Yarotsky, D. Error bounds for approximations with deep ReLU networks. *Neural Networks* **94**, 103–114 (2017).
117. Lu, Z., Pu, H., Wang, F., Hu, Z. & Wang, L. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, 6231–6239 (2017).
118. Lin, H. & Jegelka, S. ResNet with one-neuron hidden layers is a universal approximator. In *Advances in Neural Information Processing Systems*, vol. 31, 1–10 (2018).
119. Barron, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Inf. Theory* **39**, 930–945 (1993).
120. Cohen, N. & Shashua, A. Convolutional rectifier networks as generalized tensor decompositions. In *Proceedings of the International Conference on International Conference on Machine Learning*, 955–963 (2016).
121. Kumar, A., Serra, T. & Ramalingam, S. Equivalent and approximate transformations of deep neural networks. Preprint at <http://arxiv.org/abs/1905.11428> (2019).
122. DeVore, R., Hanin, B. & Petrova, G. Neural network approximation. *Acta Numer.* **30**, 327–444 (2021). **The approximation properties of NNs are described as they are presently understood and their performance with other methods of approximation is also discussed, where ReLU is centered in the analysis involving univariate and multivariate forms with both shallow and deep architectures.**

123. Huang, S.-C. & Huang, Y.-F. Bounds on the number of hidden neurons in multilayer perceptrons. *IEEE Transactions on Neural Networks* **2**, 47–55 (1991).
124. Mirchandani, G. & Cao, W. On hidden nodes for neural nets. *IEEE Transactions on Circuits Syst.* **36**, 661–664 (1989).
125. Huang, G.-B. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transactions on Neural Networks* **14**, 274–281 (2003).
126. Zhang, C., Bengio, S., Hardt, M., Recht, B. & Vinyals, O. Understanding deep learning requires rethinking generalization. In *Proceedings of the International Conference on Learning Representations*, 1–15 (2017).
127. Hardt, M. & Ma, T. Identity matters in deep learning. Preprint at <https://arxiv.org/abs/1611.04231> (2016).
128. Nguyen, Q. & Hein, M. Optimization landscape and expressivity of deep CNNs. In *Proceedings of the International Conference on Machine Learning*, vol. 80, 3730–3739 (2018).
129. Yun, C., Sra, S. & Jadbabaie, A. Generalization bounds and consistency for latent structural probit and ramp loss. In *Advances in Neural Information Processing Systems*, vol. 32 (2019).
130. Pascanu, R., Montufar, G. & Bengio, Y. On the number of response regions of deep feed forward networks with piece-wise linear activations. Preprint at <https://arxiv.org/abs/1312.6098> (2013).
131. Zaslavsky, T. *Facing up to arrangements: Face-count formulas for partitions of space by hyperplanes: Face-count formulas for partitions of space by hyperplanes*, vol. 154 (American Mathematical Society, 1975).
132. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S. & Sohl-Dickstein, J. On the expressive power of deep neural networks. In *Proceedings of the International Conference on Machine Learning*, 2847–2854 (2017).
133. Serra, T., Tjandraatmadja, C. & Ramalingam, S. Bounding and counting linear regions of deep neural networks. In *Proceedings of the International Conference on Machine Learning*, 4558–4566 (2018).
134. Hanin, B. & Rolnick, D. Complexity of linear regions in deep networks. In *Proceedings of the International Conference on Machine Learning*, 2596–2604 (2019).
135. Xiong, H. *et al.* On the number of linear regions of convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*, vol. 119, 10514–10523 (2020).
136. Goodfellow, I. J., Shlens, J. & Szegedy, C. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations* (2015).
137. Katz, G., Barrett, C., Dill, D. L., Julian, K. & Kochenderfer, M. J. ReLUpex: An efficient SMT solver for verifying deep neural networks. In *Proceedings of the International Conference on Computer Aided Verification*, 97–117 (2017).
138. Bunel, R., Turkaslan, I., Torr, P. H. S., Kohli, P. & Mudigonda, P. K. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, 4795–4804 (2018).
139. Jia, J., Cao, X., Wang, B. & Gong, N. Z. Certified robustness for top-k predictions against adversarial perturbations via randomized smoothing. In *Proceedings of the International Conference on Learning Representations* (2020).
140. Tjeng, V., Xiao, K. Y. & Tedrake, R. Evaluating robustness of neural networks with mixed integer programming. In *Proceedings of the International Conference on Learning* (2019).
141. Cheng, C.-H., Nührenberg, G. & Ruess, H. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, 251–268 (2017).
142. Wong, E. & Kolter, Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the International Conference on Machine Learning*, 5286–5295 (2018).
143. Stern, T. E. *Piecewise-linear Network Theory* (MIT Tech. Rep., 1956).
144. Katzenelson, J. An algorithm for solving nonlinear resistor networks. *The Bell Syst. Tech. J.* **44**, 1605–1620 (1965).
145. Ohtsuki, T. & Yoshida, N. Dc analysis of nonlinear networks based on generalized piecewise-linear characterization. *IEEE Transactions on Circuit Theory* **CT-18**, 146–152 (1971).
146. Chua, L. O. & Ushida, A. A switching-parameter algorithm for finding multiple solutions of nonlinear resistive circuits. *Int. J. Circuit Theory Appl.* **4**, 215–239 (1976).
147. Chien, M.-J. Piecewise-linear theory and computation of solutions of homeomorphic resistive networks. *IEEE Transactions on Circuits Syst.* **24**, 118–127 (1977).
148. Yamamura, K. & Ochiai, M. An efficient algorithm for finding all solutions of piecewise-linear resistive circuits. *IEEE Transactions on Circuits Syst.* **39**, P.213–221 (1992).
149. Pastore, S. & Premoli, A. Polyhedral elements: A new algorithm for capturing all the equilibrium points of piecewise-linear circuits. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **40**, 124–132 (1993).
150. Yamamura, K. & Ohshima, T. Finding all solutions of piecewise-linear resistive circuits using linear program-

- ming. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **45**, 434–445 (1998).
151. Chua, L. O. Modeling of three terminal devices: A black box approach. *IEEE Transactions on Circuit Theory* **19**, 555–562 (1972).
 152. Meijer, P. B. Fast and smooth highly nonlinear multi-dimensional table models for device modeling. *IEEE Transactions on Circuits Syst.* **37**, 335–346 (1990).
 153. Yamamura, K. On piecewise-linear approximation of nonlinear mappings containing gummel-poon models or schichman-hodges models. *IEEE Transactions on Circuits Syst. I: Fundamental Theory Appl.* **39**, 694–697 (1992).
 154. Chua, L. O., Komuro, M. & Matsumoto, T. The double scroll family. *IEEE Transactions on Circuits Syst.* **33**, 1072–1118 (1986).
 155. Billings, S. & Voon, W. Piecewise linear identification of non-linear systems. *Int. J. Control.* **46**, 215–235 (1987).
 156. Sontag, E. From linear to nonlinear: some complexity comparisons. In *Proceedings of the IEEE Conference on Decision and Control*, vol. 3, 2916–2920 (1995).
 157. Mestl, T., Plahte, E. & Omholt, S. W. Periodic solutions in systems of piecewise-linear differential equations. *Dyn. & Stab. Syst.* **10**, 179–193 (1995).
 158. Yalcin, M., Suykens, J. A. & Vandewalle, J. *Cellular neural networks, multi-scroll chaos and synchronization*, vol. 50 (World Scientific, 2005).
 159. Yu, J., Mu, X., Xi, X. & Wang, S. A memristor model with piecewise window function. *Radioengineering* **22**, 969–974 (2013).
 160. Mu, X., Yu, J. & Wang, S. Modeling the memristor with piecewise linear function. *Int. J. Numer. Model. Electron. Networks Devices & Fields* **28**, 96–106 (2015).
 161. Yu, Y., Juntang Li, Mu, X., Zhang, J., Miao, X. & Wang, S. Modeling the AgInSbTe memristor. *Radioengineering* **24**, 808–813 (2015).
 162. Yu, J. *Memristor model with window function and its applications*. Ph.D. thesis, Tsinghua University (2016).
 163. Bemporad, A., Torrisi, F. D. & Morari, M. Optimization-based verification and stability characterization of piecewise affine and hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, 45–58 (2000).
 164. Bemporad, A., Ferrari-Trecate, G. & Morari, M. Observability and controllability of piecewise affine and hybrid systems. *IEEE transactions on automatic control* **45**, 1864–1876 (2000).
 165. Heemels, W., De Schutter, B. & Bemporad, A. Equivalence of hybrid dynamical models. *Automatica* **37**, 1085–1091 (2001).
 166. Bemporad, A. Piecewise linear regression and classification. Preprint at <https://arxiv.org/abs/2103.06189> (2021).
 167. Huang, X., Xu, J. & Wang, S. Nonlinear system identification with continuous piecewise linear neural network. *Neurocomputing* **77**, 167–177 (2012).
 168. Huang, X., Mu, X. & Wang, S. Continuous piecewise linear identification with moderate number of subregions. In *the 16th IFAC Symposium on System Identification*, 535–540 (2012).
 169. Tao, Q. *et al.* Short-term traffic flow prediction based on the efficient hinging hyperplanes neural network. *IEEE Transactions on Intell. Transp. Syst.* 1–13 (2022).
 170. Pistikopoulos, E. N., Dua, V., Bozinis, N. A., Bemporad, A. & Morari, M. On-line optimization via off-line parametric optimization tools. *Comput. & Chem. Eng.* **26**, 175–185 (2002).
 171. Bemporad, A., Borrelli, F. & Morari, M. Piecewise linear optimal controllers for hybrid systems. In *Proceedings of the American Control Conference*, vol. 2, 1190–1194 (2000). **The characteristic of PWL in control systems and the applications of PWL nonlinearity are introduced.**
 172. Bemporad, A., Borrelli, F. & Morari, M. Model predictive control based on linear programming - the explicit solution. *IEEE Transactions on Autom. Control.* **47**, 1974–1985 (2002).
 173. Bemporad, A., Morari, M., Dua, V. & Pistikopoulos, E. N. The explicit linear quadratic regulator for constrained systems. *Automatica* **38**, 3–20 (2002).
 174. Chikkula, Y., Lee, J. & Okunnaik, B. Dynamically scheduled model predictive control using hinging hyperplane models. *AIChE J.* **44**, 2658–2674 (1998).
 175. Wen, C., Ma, X. & Ydstie, B. E. Analytical expression of explicit mpc solution via lattice piecewise-affine function. *Automatica* **45**, 910–917 (2009).
 176. Xu, J. & Wang, S. Lattice piecewise affine representations on convex projection regions. In *Proceedings of the IEEE Conference on Decision and Control*, 7240–7245 (2019).
 177. Yue-Hei Ng, J. *et al.* Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4694–4702 (2015).
 178. Purwins, H. *et al.* Deep learning for audio signal processing. *IEEE J. Sel. Top. Signal Process.* **13**, 206–219 (2019).
 179. Xie, Q., Luong, M.-T., Hovy, E. & Le, Q. V. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10687–10698 (2020).

180. Qiao, Y. *et al.* Fpga-accelerated deep convolutional neural networks for high throughput and energy efficiency. *Concurr. Comput. Pract. Exp.* **29**, e3850 (2017).
181. Dua, D. & Graff, C. UCI machine learning repository. <http://archive.ics.uci.edu/ml> (2017).
182. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. *et al.* Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998). **The basic learning framework for generic DNNs including PWL-DNNs is formally introduced in this work.**
183. Netzer, Y. *et al.* Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011* (2011).
184. LeCun, Y., Huang, F. J. & Bottou, L. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, II–104 (2004).
185. Krizhevsky, A. & Hinton, G. *Learning multiple layers of features from tiny images* (Technical report, University of Toronto, 2009).
186. Lin, T.-Y. *et al.* Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision*, 740–755 (2014).
187. Russakovsky, O. *et al.* ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **115**, 211–252 (2015).
188. Krishna, R. *et al.* Visual genome: Connecting language and vision using crowdsourced dense image annotations. *Int. J. Comput. Vis.* **123**, 32–73 (2017).
189. Abadi, M. *et al.* TensorFlow: Large-scale machine learning on heterogeneous systems. <https://www.tensorflow.org/> (2015).
190. Chollet, F. Keras. <https://github.com/fchollet/keras> (2015).
191. Jia, Y. *et al.* Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM international conference on Multimedia*, 675–678 (2014).
192. Chen, T. *et al.* MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. Preprint at <https://arxiv.org/abs/1512.01274> (2015).
193. Bergstra, J. *et al.* Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference* (2010).
194. Tao, Q. *et al.* Toward deep adaptive hinging hyperplanes. *IEEE Transactions on Neural Networks Learn. Syst.* (2021).
195. Tang, C. *et al.* Sparse MLP for image recognition: Is self-attention really necessary? Preprint at <https://arxiv.org/abs/2109.05422> (2021).
196. Wang, Y., Li, Z., Xu, J. & Li, J. Multilevel lattice piecewise linear representation and its application in explicit predictive control. In *Proceedings of the Asian Control Conference*, 1066–1071 (2019).
197. Kawaguchi, K. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, 586–594 (2016).
198. Yun, C., Sra, S. & Jadbabaie, A. Global optimality conditions for deep neural networks. Preprint at <https://arxiv.org/abs/1707.02444> (2017).
199. Nguyen, Q. & Hein, M. The loss surface of deep and wide neural networks. In *Proceedings of the International Conference on Machine Learning*, vol. 70, 2603–2612 (2017).
200. Yun, C., Sra, S. & Jadbabaie, A. Small nonlinearities in activation functions create bad local minima in neural networks. In *Proceedings of the International Conference on Learning Representations* (2019).