Journal of Global Optimization
https://doi.org/10.1007/s10898-023-01344-2

DC-programming for neural network optimizations

Pranjal Awasthi[1] · Anqi Mao[2] · Mehryar Mohri[1] · Yutao Zhong[2]

Received: 10 September 2022 / Accepted: 12 November 2023

Abstract
We discuss two key problems related to learning and optimization of neural networks: the computation of the adversarial attack for adversarial robustness and approximate optimization of complex functions. We show that both problems can be cast as instances of DC-programming. We give an explicit decomposition of the corresponding functions as differences of convex functions (DC) and report the results of experiments demonstrating the effectiveness of the DCA algorithm applied to these problems.

Keywords Neural networks · Adversarial robustness · Function approximation · Optimization · DC-programming

1 Motivation

Optimization problems based on neural networks arise in a variety of different contexts, including function approximation, pattern recognition, sequential decision making, data compression and many others. This work is motivated by two key motivations highlighted below.
Adversarial attacks for adversarial robustness. Neural networks trained on large datasets have achieved breakthroughs in speech and visual recognition tasks and many other applications in recent years [24, 54]. However, these models have been shown to be susceptible to imperceptible perturbations [55]. This motivates the study of *adversarial robustness*, that is, the design of classifiers that are robust to perturbations within a small ball around the input, typically measured in the ℓ_p norm [5, 7, 9, 15, 37, 40, 57].

✉ Yutao Zhong
yutao@cims.nyu.edu

Pranjal Awasthi
pranjalawasthi@google.com

Anqi Mao
aqmao@cims.nyu.edu

Mehryar Mohri
mohri@google.com

1 Google Research, New York, NY 10011, USA

2 Courant Institute, New York, NY 10012, USA

Thus, in the adversarial setting, the standard binary classification loss is replaced by its adversarial counterpart, which is defined, for a given data point, as the supremum of the binary loss over a ball of a pre-specified radius around the given point. The adversarial counterpart of the classification margin loss is similarly defined via its supremum over the ball. Optimizing the binary loss or adversarial binary loss directly is NP-hard for most hypothesis sets. A natural alternative, instead, is to optimize a surrogate margin-based loss with an alternative function Φ, such as the hinge loss, that admits more favorable properties such as differentiability, convexity and statistical consistency with respect to the target loss [2–4, 8, 29, 36, 38, 39, 41, 42, 53, 56, 62, 63].

However, the evaluation of the above surrogates still requires computing the *adversarial attack*, that is solving an optimization problem related to the presence of the supremum in the definition of the loss. For a surrogate margin-based loss, that amounts to a *margin minimization problem*, that is finding the strongest adversarial attack point in the ball that minimizes the margin—this objective should of course be distinguished from the standard goal of margin-maximization in learning and generalization [45].

Margin minimization is a notoriously difficult problem in adversarial training for which a variety of heuristics have been developed. We will show that this problem can be cast as a DC-programming problem by proving a decomposition of the multi-class margin as a difference of two convex functions. The DCA algorithm of Pham Dinh and Le Thi [46, 47] can then be used to solve the optimization problem.

We note that Seck et al. [50] also applied DCA to adversarial robustness verification. However, their optimization problem is to find the minimal perturbation required to flip the label of the clean example, which is distinct from our setting. In addition, the approaches used to address the problems are different: their technique involved the use of DC to eliminate some binary variables in the optimization problem, while our method provides an explicit DC-decomposition of neural networks to solve the margin minimization problem.

Approximate optimization of complex functions. Neural networks with at least one hidden-layer and enough hidden units are known to be universal approximators, that is they are able to approximate any measurable function defined over finite-dimensional spaces to an arbitrary degree of accuracy [22]. On the other hand, minimization or maximization of an arbitrary complex function still remains an open problem in the optimization literature. The expressive power of neural networks and the challenge of optimizing an arbitrary function motivates the study of *approximate optimization*, that is, the design of algorithms that can closely optimize any function that is well approximated by neural networks.

Let F be a very complex or costly function to optimize. Then, the approximate optimization procedure consists of the following steps:

– Sampling: draw a large number of pairs $\{*\}(x^i, F(x^i))_{i=1}^m$;
– Learning: use a deep neural network h to fit F on that sample by minimizing an appropriate loss;
– Optimization: find the minimizer or maximizer of the learned neural network h.

The first two steps of sampling and learning have been extensively studied in the literature. However, the final step of optimization of neural networks brings up new challenges due to the lack of convexity, the complexity of architectures and the diversity of neural networks. We will show that the neural networks typically used in practice are all DC-functions (difference of convex functions) of the input feature vector and thus that the DCA algorithm [46, 47] can be adopted in the final step, which helps solve efficiently the approximate optimization problem.

In the next section, we introduce the notation and some basic concepts for DC-programming and the hypothesis sets of neural networks considered in this paper: multilayer perceptrons (MLPs) and convolutional neural networks (CNNs). In Sect. 3, we derive the explicit form of a DC-decomposition for MLPs and CNNs, which we use to derive a DCA solution for the approximate optimization of complex functions. In Sect. 4, we give a detailed description of the margin minimization problem and present a DCA solution for the corresponding computation of the adversarial attack for adversarial robustness. To do so, we prove that the DC-decomposition derived in Sect. 3 naturally leads to a DC-decomposition of the margin. In Sect. 5, we report the results of experiments demonstrating the effectiveness of the DCA algorithms proposed in Sects. 3 and 4.

2 Preliminaries

We start by introducing some basic concepts and results that are related to DC-programming.

2.1 DC-programming definitions and algorithms

Definition 1 (DC-functions) We say that a hypothesis $h\colon \mathbb{R}^n \to \mathbb{R}$ is a *DC-function* if it admits a *DC-decomposition*, that is, if it can be written as the difference of two convex functions (DC) f and g:

$$h = f - g.$$

In that case, functions f and g are called *DC-components* of the hypothesis h.

The optimization problem corresponding to such a function h is referred to as a *DC-programming* problem and defined as follows:

$$p := \inf \left\{ h(x) = f(x) - g(x) \colon x \in \mathbb{R}^n \right\}. \tag{1}$$

Observe that any constraint $x \in \mathcal{C}$ with a closed convex set $\mathcal{C} \subset \mathbb{R}^n$ can be equivalently incorporated into the standard DC-program by letting $h(x) = [f(x) + \chi_{\mathcal{C}}(x)] - g(x)$, where $\chi_{\mathcal{C}}(x) := \begin{cases} 0 & x \in \mathcal{C} \\ +\infty & \text{otherwise.} \end{cases}$

To find the global minimum of (1), there have been several approaches discussed in the literature [48], including the pioneering combinatorial approach proposed by Hoang [17], its further developed solution for low-rank non-convex structures [16], and the branch-and-bound algorithm which has an exponential convergence Reiner and N. V., [49] with the correction of Hoang [18]. Nevertheless, as pointed by Pham Dinh and Le Thi [46], these global algorithms are not able to solve real-world high-dimensional DC-programs.

Instead, an alternative method based on convex analysis, *the DCA algorithm* [46, 47], is often adopted in practice, which can be further combined with branch-and-bound techniques to find a global optimum. When the function g is sub-differentiable, DCA coincides with the concave-convex procedure (CCCP) of Yuille and Rangarajan [61]. The pseudocode of DCA is given for that case in Algorithm 1; here, ∂g denotes any subgradient of the function g and $\cdot$ denotes the inner product of two vectors.

Algorithm 1 DCA algorithm solving DC-program (1).

```
input: Maximum number of iterations $T$, initial point $x_1$, tolerances $\epsilon_1$ and $\epsilon_2$
for each round $t = 1, \dots, T$ do
    $x_{t+1} \in \operatorname{argmin}\{*\}\mathcal{C}_t(x) := f(x) - g(x_t) - \partial g(x_t) \cdot (x - x_t) : x \in \mathbb{R}^n$
    if $|(f - g)(x_{t+1}) - (f - g)(x_t)| \leq \epsilon_1$ or $\| * \|x_{t+1} - x_t \leq \epsilon_2$ then
        break
    end if
end for
```

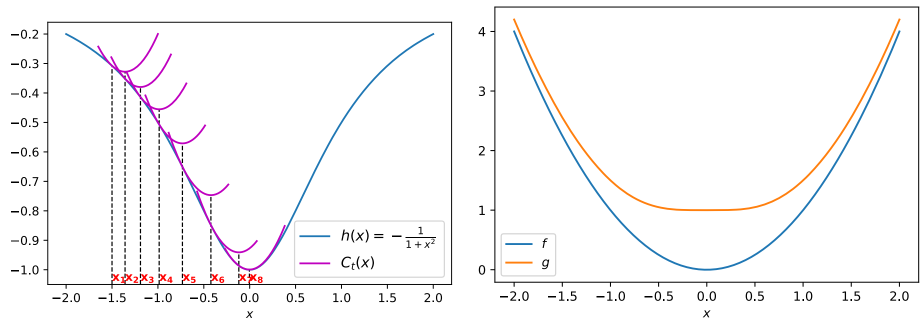

Fig. 1 Left: an illustration of DCA in Algorithm 1 for minimizing the non-convex function $h : x \mapsto -\frac{1}{1+x^2}$. For each t, the convex function $\mathcal{C}_t$ has the same value and derivative as h at $x = x_t$ and upper bounds h by definition. Right: DC-components f and g of the function $h : x \mapsto -\frac{1}{1+x^2}$, where $f : x \mapsto x^2$ and $g : x \mapsto x^2 + \frac{1}{1+x^2}$

The algorithm consists of iteratively replacing g with its first-order approximation and solving the resulting convex optimization problem. The stopping criterion is checked at the end of each iteration. Figure 1 illustrates this procedure for minimizing the non-convex function $h : x \mapsto -\frac{1}{1+x^2}$ that can be decomposed into the difference of two convex functions f and g. DCA is a primal-dual sub-differential method [46, 47], which can handle DC-programs (1) with proper lower semi-continuous convex functions f and g. In this paper, we propose solutions based on DCA (Algorithm 2 and Algorithm 3) for the computation of the adversarial attack and approximate optimization problems described in Sect. 1 and further demonstrate their effectiveness for these problems in Sect. 5.

The DCA algorithm is based on the decomposition of a function, and benefits from the flexibility of such a decomposition. Indeed, if $h = f - g$ is a DC-decomposition, adding a convex auxiliary function to both f and g still results in a DC-decomposition. In practice, a good auxiliary function such as one admitting strongly-convexity, e.g. $\lambda\| \cdot \|^2$ often makes the DCA algorithm more efficient [33, 34]. The convergence of DCA has been discussed extensively in the literature [31, 46, 47, 51, 59]. Pham Dinh and Le Thi [46] first proved that DCA is guaranteed to converge to a critical point. The same result also holds for the CCCP algorithm, which is a special case of DCA [51]. Moreover, DCA can find the global optimum of the trust region problem [47]. For many DC-programs related to support vector machines (SVM) [10], the DCA algorithm admits linear convergence and thus its number of iterations is relatively small [59]. In practice, an effective heuristic adopted in many applications consists of using DCA with multiple restarting points to minimize the objective function [43, 44]. In some applications, global optimality can be efficiently tested and in fact DCA typically leads to the global optimum without even resorting to such heuristics [11, 19–21].

DCA can also often be combined with branch-and-bound techniques to determine the global optimum, which can be achieved efficiently in some instances [30, 35]. More generally, as pointed out by Le Thi and Pham Dinh [32–34], DCA admits the following benefits: (1) Flexibility: a suitable choice of the DC-decomposition can make DCA more robust and efficient, and even lead to the global optimum; (2) Convergence: DCA admits linear convergence for general DC-programs and, in particular, admits finite convergence for polyhedral DC-programs; (3) Versatility: DCA can recover many standard algorithms with a careful choice of the DC-decomposition for both convex and non-convex optimization. For a convex program, DCA can converge to the global optimum by reinterpreting it as a DC-program.

In view of these advantages, DCA can be applied to a wide range of non-convex optimization problems and applications including non-convex quadratic programs [13], the trust region problem [47], kernel selection [1], learning in second-price reserve auctions [43, 44], forecasting time series [26–28], discrepancy estimation in domain adaptation [6], eigenvalue problems [52] and many others, where DCA is applicable to large-scale scenarios, adapted to flexible target needs, and benefitting from theoretical guarantees. In this paper, we will apply DCA to two problems related to learning and optimization of neural networks.

2.2 Neural network definitions

Let $\mathcal{X}$ denote the input feature space and $\mathcal{Y}$ denote the label space. In the approximate optimization of complex function, $\mathcal{Y} = \mathbb{R}$ is real-valued, while in adversarial robustness, the computation of the adversarial attack, $\mathcal{Y} = \{1, \ldots, c\}$ is a set of $c \geq 2$ classes. Let $\mathcal{H}$ be a hypothesis set of functions mapping from $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$ and $\overline{h}(x) = (h(x, 1), \ldots, h(x, c))$ be the output vector of a hypothesis $h \in \mathcal{H}$ in multi-class classification. For each class $z \in \{1, \ldots, c\}$, real-valued $h(x, z)$ can be viewed as the score assigned to class z by h. For example, if we let $\mathcal{H}$ be the family of feedforward neural networks with L hidden layers (will be introduced soon in (2)), then, $\overline{h}(x)$ is equal to $a^{[L+2]}(x)$ and $\{x \mapsto h(x, z),\ z \in \{1, \ldots, c\}\}$ are the component functions of $a^{[L+2]}$. We denote by $\ell\colon \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ a loss function and thus by $\ell(h, x, y)$ the loss of a hypothesis h for a pair (x, y). For the hypothesis set of neural networks, we will specifically consider the family of feedforward neural networks (also known as multilayer perceptrons (MLPs)) and convolutional neural networks (CNNs) [14].

The feedforward neural network is a quintessential artificial neural network, which typically consists of one input layer, L hidden layers and one output layer, and can be represented in the following form:

$$\begin{aligned} a^{[1]}(x) &= x \in \mathbb{R}^{n_1}, \\ a^{[l]}(x) &= \sigma\left(W^{[l]} a^{[l-1]}(x) + b^{[l]}\right) \in \mathbb{R}^{n_l}, \text{ for } l = 2, \ldots, L+1, \\ a^{[L+2]}(x) &= W^{[L+2]} a^{[L+1]}(x) + b^{[L+2]} \in \mathbb{R}^{n_{L+2}}, \end{aligned} \tag{2}$$

where given an input $x \in \mathbb{R}^{n_1}$, we use n_l to denote the dimension of the output of the lth layer, $a^{[l]}(x)$. Here, $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $b^{[l]} \in \mathbb{R}^{n_l}$ denote the weight matrix and the offset vector at layer l, and σ denotes the ReLU activation. The expressions (2) define vector-valued functions $a^{[l]}\colon \mathbb{R}^{n_1} \to \mathbb{R}^{n_l}$, for $l = 1, \ldots, L+2$. For the MLP given by (2), we also refer to $L+2$ as its depth and n_l as the number of units at the lth layer. The simplest MLPs are those with one hidden layer, and often referred to as one-hidden-layer neural networks, i.e., in the form of (2) with $L = 1$.

The convolutional neural network is a specialized kind of neural networks for processing input feature vector with a known grid-like structure, such as images with a two-dimensional grid of pixels. The CNN typically has convolution layers and pooling layers. Let X denote the input image with height n_{h_1}, width n_{w_1} and the number of channels n_{c_1}. The convolution layers can be represented in the following form:

$$\begin{aligned} &s^{[1]}(X) = X \in \mathbb{R}^{n_{h_1} \times n_{w_1} \times n_{c_1}}, \\ &s^{[l]}(X) = \sigma\big(W^{[l]} * s^{[l-1]}(X) + b^{[l]}\big) \in \mathbb{R}^{n_{h_l} \times n_{w_l} \times n_{c_l}}, \ \text{for } l = 2, \dots, L+1, \end{aligned} \tag{3}$$

where given an input image $X \in \mathbb{R}^{n_{h_1} \times n_{w_1} \times n_{c_1}}$, $s^{[l]}(X)$ is the output of the lth convolution layer with dimension $n_{h_l} \times n_{w_l} \times n_{c_l}$, for $l = 2, \dots, L+1$. Here, σ is the ReLU activation, $W^{[l]} \in \mathbb{R}^{k_{h_l} \times k_{w_l} \times n_{c_{l-1}} \times n_{c_l}}$ is the convolution kernel and $b^{[l]} \in \mathbb{R}^{1 \times 1 \times 1 \times n_{c_l}}$ is the offset at layer l. The symbol $*$ is used to denote the convolution. With VALID padding and a stride of 1×1 [14], the result of the convolution $W^{[l]} * s^{[l-1]}(X)$ is of the shape $n_{h_l} \times n_{w_l} \times n_{c_l}$, where

$$\begin{aligned} n_{h_l} &= n_{h_{l-1}} - k_{h_l} + 1, \\ n_{w_l} &= n_{w_{l-1}} - k_{w_l} + 1, \end{aligned}$$

and it is defined by the following equation:

$$\big(W^{[l]} * s^{[l-1]}(X)\big)_{i,j,t} = \sum_{r=0}^{n_{c_{l-1}}-1} \sum_{u=0}^{k_{h_l}-1} \sum_{v=0}^{k_{w_l}-1} W^{[l]}_{u,v,r,t} s^{[l-1]}(X)_{i+u,j+v,r}.$$

The pooling layers further modify the output by replacing a certain element with summary statistics of the elements within its neighborhood. We will consider two kinds of pooling: max-pooling and average pooling, which report the maximum output and average output within a rectangular neighborhood respectively [14].

3 DC-decomposition of neural networks

In this section, we prove that many commonly used types of neural networks are DC-functions of the input feature vector and provide an explicit DC-decomposition for these functions. Building upon that, we present a DCA solution for the approximate optimization of complex functions.

3.1 One-hidden-layer neural networks

We start the analysis with one-hidden-layer neural networks, which admit the following form:

$$\begin{aligned} a^{[1]}(x) &= x \in \mathbb{R}^{n_1}, \\ a^{[2]}(x) &= \sigma\big(W^{[2]} a^{[1]}(x) + b^{[2]}\big) \in \mathbb{R}^{n_2}, \\ a^{[3]}(x) &= W^{[3]} a^{[2]}(x) + b^{[3]} \in \mathbb{R}^{n_3}. \end{aligned} \tag{4}$$

Here, given an input $x \in \mathbb{R}^{n_1}$, the vectors $a^{[1]}(x)$, $a^{[2]}(x)$, and $a^{[3]}(x)$ are the outputs of the input layer, the hidden layer and the output layer with dimension n_1, n_2, and n_3 respectively. The next theorem shows that each component function of the vector-valued function $a^{[3]}: \mathbb{R}^{n_1} \to \mathbb{R}^{n_3}$ is a DC-function of x.

Theorem 1 *For one-hidden-layer neural networks (4), the function $a_i^{[3]}$ can be written as $f_i^{[3]} - g_i^{[3]}$, for $i = 1, \dots, n_3$, where $f_i^{[3]}$ and $g_i^{[3]}$ are convex in x.*

Proof Since the composition with affine function preserves convexity, any component function of $a^{[2]}$ is convex in x. Note that the weight matrix $W^{[3]}$ can be expressed in terms of its positive and negative parts as

$$W^{[3]} = W_+^{[3]} - W_-^{[3]},$$

where $W_+^{[3]}$ and $W_-^{[3]}$ are both nonnegative. Therefore, $a^{[3]}$ admits the following decomposition:

$$a^{[3]} = \left(W_+^{[3]} a^{[2]} + b^{[3]}\right) - \left(W_-^{[3]} a^{[2]}\right).$$

We can let $f^{[3]} = W_+^{[3]} a^{[2]} + b^{[3]}$ and $g^{[3]} = W_-^{[3]} a^{[2]}$, since nonnegative weighted sum preserves convexity. □

Theorem 1 also holds for the special case where $a^{[3]}$ only has one component function. This implies that any one-hidden-layer neural network (4) with $n_3 = 1$ admits a DC-decomposition.

Corollary 1 *Any one-hidden-layer neural network with one output unit is a DC-function of the input feature vector.*

3.2 Multi-layer perceptrons

The decomposition of one-hidden-layer neural networks can be extended to multilayer perceptrons with L hidden layers (2). The next theorem shows that for an MLP, each component function of the vector-valued function $a^{[l]} : \mathbb{R}^{n_1} \to \mathbb{R}^{n_l}$, for $l = 1, \dots, L+2$, is a DC-function of x, which generalizes Theorem 1.

Theorem 2 *For multi-layer perceptrons (2), any component function of the vector-valued function $a^{[l]}$, for $l = 1, \dots, L+2$, is a DC-function of the input feature vector x.*

Proof Clearly, any component function of $a^{[1]}$ is convex and thus is a DC-function of x. Since the composition with affine function preserves convexity, any component function of $a^{[2]}$ is convex and thus is a DC-function of x as well.

We then proceed by induction on l. Assume that $a^{[l]}$ admits the following decomposition:

$$a^{[l]} = f^{[l]} - g^{[l]},$$

where the component functions of $f^{[l]}$ and $g^{[l]}$ are all convex in x. Then, $c^{[l+1]} := W^{[l+1]} a^{[l]} + b^{[l+1]}$ can be written as

$$\begin{aligned} c^{[l+1]} &= \left(W_+^{[l+1]} - W_-^{[l+1]}\right)\left(f^{[l]} - g^{[l]}\right) + b^{[l+1]} \\ &= \left(W_+^{[l+1]} f^{[l]} + W_-^{[l+1]} g^{[l]} + b^{[l+1]}\right) - \left(W_-^{[l+1]} f^{[l]} + W_+^{[l+1]} g^{[l]}\right). \end{aligned}$$

Since non-negative weighted sum preserves convexity, $\tilde{f}^{[l+1]} := W_+^{[l+1]} f^{[l]} + W_-^{[l+1]} g^{[l]} + b^{[l+1]}$ and $\tilde{g}^{[l+1]} := W_-^{[l+1]} f^{[l]} + W_+^{[l+1]} g^{[l]}$ both have convex component functions. Hence, the component functions of $c^{[l+1]}$ are DC-functions. Now, we have

$$a^{[l+1]} = \sigma\left(\tilde{f}^{[l+1]} - \tilde{g}^{[l+1]}\right)$$

$$= \max(\tilde{f}^{[l+1]}, \tilde{g}^{[l+1]}) - \tilde{g}^{[l+1]},$$

where the component functions of $f^{[l+1]} := \max(\tilde{f}^{[l+1]}, \tilde{g}^{[l+1]})$ and $g^{[l+1]} := \tilde{g}^{[l+1]}$ are all convex.

Note that $a^{[L+2]} = c^{[L+2]}$. Therefore, any component function of $a^{[l]}$, for $l = 1, \ldots, L+2$, is a DC-function. □

Similarly to Theorem 1, Theorem 2 holds for the special case where $a^{[L+2]}$ only has one component function. Thus, Theorem 2 implies that any multilayer perceptron (2) with $n_{L+2} = 1$ admits a DC-decomposition.

Corollary 2 *Any multilayer perceptron with one output unit is a DC-function of the input feature vector.*

The multilayer perceptrons with one output unit are useful for the approximation optimization of real-valued complex functions, as indicated in Sect. 1. Our results like Corollary 1 and Corollary 2 are thus helpful for the design of DCA solution to these problems as shown in Sect. 3.4.

3.3 Convolutional neural networks (CNNs)

As with dense layers, DC-decomposition also works for convolution layers (3). Theorem 3 shows that DC-decomposition can be constructed for any component function of the function $s^{[l]}$ at convolution layer l.

Theorem 3 *For convolution layers (3), any component function of $s^{[l]}$, for $l = 1, \ldots, L+1$, is a DC-function of the input X.*

Proof Clearly, any component function of $s^{[1]}$ is convex and thus is a DC-function of X. Since the composition with affine function preserves convexity, any component function of $s^{[2]}$ is also convex and thus is a DC-function of X.

We then proceed with induction on l. Assume that $s^{[l]}$ admits a decomposition

$$s^{[l]} = f^{[l]} - g^{[l]},$$

where the component functions of $f^{[l]}$ and $g^{[l]}$ are all convex in X. Then, $c^{[l+1]} := W^{[l+1]} * a^{[l]} + b^{[l+1]}$ can be written as

$$\begin{aligned} c^{[l+1]} &= \left(W_+^{[l+1]} - W_-^{[l+1]}\right) * \left(f^{[l]} - g^{[l]}\right) + b^{[l+1]} \\ &= \left(W_+^{[l+1]} * f^{[l]} + W_-^{[l+1]} * g^{[l]} + b^{[l+1]}\right) - \left(W_-^{[l+1]} * f^{[l]} + W_+^{[l+1]} * g^{[l]}\right). \end{aligned}$$

Since non-negative weighted sum preserves convexity, $\tilde{f}^{[l+1]} := W_+^{[l+1]} * f^{[l]} + W_-^{[l+1]} * g^{[l]} + b^{[l+1]}$ and $\tilde{g}^{[l+1]} := W_-^{[l+1]} * f^{[l]} + W_+^{[l+1]} * g^{[l]}$ both have convex component functions. Hence, the component functions of $c^{[l+1]}$ are DC-functions. Now we have

$$\begin{aligned} s^{[l+1]} &= \sigma\left(\tilde{f}^{[l+1]} - \tilde{g}^{[l+1]}\right) \\ &= \max(\tilde{f}^{[l+1]}, \tilde{g}^{[l+1]}) - \tilde{g}^{[l+1]}, \end{aligned}$$

where the component functions of $f^{[l+1]} := \max(\tilde{f}^{[l+1]}, \tilde{g}^{[l+1]})$ and $g^{[l+1]} := \tilde{g}^{[l+1]}$ are all convex.

Therefore, any component function of $s^{[l]}$, for $l = 1, \ldots, L+1$, is a DC-function. □

DC-decomposition is also compatible with pooling layers. We give the analysis for both average pooling and max pooling as follows.
Average pooling. Let $\mathcal{A}$ denote the average pooling operation. Theorem 4 shows that the DC structure is preserved by the average pooling.

Theorem 4 *If all the component functions of $s^{[l]}$ are DC-functions of the input x, then the component functions of $\mathcal{A} \circ s^{[l]}$ are also DC-functions of x.*

Proof Suppose that $s^{[l]}$ admits a decomposition

$$s^{[l]} = f^{[l]} - g^{[l]},$$

where the component functions of $f^{[l]}$ and $g^{[l]}$ are all convex in x. Since the average pooling operation is linear, $\mathcal{A} \circ s^{[l]}$ can be written as

$$\mathcal{A} \circ s^{[l]} = \mathcal{A} \circ f^{[l]} - \mathcal{A} \circ g^{[l]}.$$

Since non-negative weighted sum preserves convexity, the component functions of $\mathcal{A} \circ f^{[l]}$ and $\mathcal{A} \circ g^{[l]}$ are all convex in x. □

Max pooling. Let $\mathcal{M}$ denote the max pooling operation. Theorem 5 shows that the DC structure is preserved by the max pooling.

Theorem 5 *If all the component functions of $s^{[l]}$ are DC-functions of the input x, then the component functions of $\mathcal{M} \circ s^{[l]}$ are also DC-functions of x.*

Proof Suppose that $s^{[l]}$ admits a decomposition

$$s^{[l]} = f^{[l]} - g^{[l]},$$

where the component functions of $f^{[l]}$ and $g^{[l]}$ are all convex in x. Assume that the kernel size of the max pooling layer is $k_h \times k_w$. Then, $\mathcal{M} \circ s^{[l]}$ can be written as

$$\mathcal{M} \circ s^{[l]} = \left(\mathcal{M} \circ s^{[l]} + k_h k_w \mathcal{A} \circ g^{[l]}\right) - k_h k_w \mathcal{A} \circ g^{[l]}.$$

Since non-negative weighted sum preserves convexity, the component functions of $k_h k_w \mathcal{A} \circ g^{[l]}$ are all convex in x. The component functions of $\mathcal{M} \circ s^{[l]} + k_h k_w \mathcal{A} \circ g^{[l]}$ are all convex in x as well. Indeed, each one of them is of the following form:

$$\begin{aligned}
&\max_{\substack{0 \le u \le k_h - 1 \\ 0 \le v \le k_w - 1}} \{ f^{[l]}_{i+u,j+v,r} - g^{[l]}_{i+u,j+v,r} \} + k_h k_w \frac{1}{k_h k_w} \sum_{u'=0}^{k_h-1} \sum_{v'=0}^{k_w-1} g^{[l]}_{i+u',j+v',r} \\
&= \max_{\substack{0 \le u \le k_h - 1 \\ 0 \le v \le k_w - 1}} \{ f^{[l]}_{i+u,j+v,r} + \sum_{u'=0}^{k_h-1} \sum_{v'=0}^{k_w-1} g^{[l]}_{i+u',j+v',r} - g^{[l]}_{i+u,j+v,r} \},
\end{aligned}$$

for some i, j, r. It is convex in x since sum and pointwise maximum preserve convexity. □

Theorem 3, Theorem 4 and Theorem 5 imply that any convolutional neural network with one output unit admits a DC-decomposition.

Corollary 3 *Any convolutional neural network with one output unit is a DC-function of the input X.*

Algorithm 2 DCA solution for the approximate optimization of complex functions.

```
input: Maximum number of iterations $T$, sample $\{*\}(x^i, F(x^i))_{i=1}^m$, loss function $\ell$, initial point $x_1$, tolerances $\epsilon_1$ and $\epsilon_2$
learn neural networks: $h_{\mathrm{NN-aprox}} \in \mathrm{argmin}_{h \in \mathcal{H}_{\mathrm{NN-aprox}}} \sum_{i=1}^m \ell(h, x^i, F(x^i))$
decompose by Section 3: $h_{\mathrm{NN-aprox}} = f_{\mathrm{NN-aprox}} - g_{\mathrm{NN-aprox}}$
for each round $t = 1, \ldots, T$ do
    $x_{t+1} \in \mathrm{argmin}\{*\} f_{\mathrm{NN-aprox}}(x) - g_{\mathrm{NN-aprox}}(x_t) - \partial g_{\mathrm{NN-aprox}}(x_t) \cdot (x - x_t) \colon x \in \mathbb{R}^n$
    if $|(f_{\mathrm{NN-aprox}} - g_{\mathrm{NN-aprox}})(x_{t+1}) - (f_{\mathrm{NN-aprox}} - g_{\mathrm{NN-aprox}})(x_t)| \leq \epsilon_1$ or $\| * \| x_{t+1} - x_t \leq \epsilon_2$
    then
        break
    end if
end for
```

As with the multilayer perceptrons, convolutional neural networks with one output unit can also be used for the approximation optimization of real-valued complex functions. Thus, Corollary 3 can serve as a tool for the design of a DCA solution for approximation optimization problems with CNNs. Furthermore, we will see that the results such as Corollary 3 are also helpful for the design of a DCA solution for the margin minimization in adversarial robustness where convolutional neural networks are used as common hypotheses, as detailed in Sect. 4 and Sect. 5.

3.4 DCA solution for approximate optimization

Section 3 shows that there exists a DC-decomposition for the feedforward neural networks and convolutional neural networks. Building upon this, we can give our DCA solution for the approximate optimization problem introduced in Sect. 1 using such neural networks with one output unit, denoted as $\mathcal{H}_{\mathrm{NN-aprox}}$. The pseudocode of our algorithm is given in Algorithm 2.

In Sect. 5, we also report the empirical results which further demonstrate the effectiveness of our Algorithm 2.

4 DC-decomposition of confidence margin

In this section, we show that the margin of a hypothesis h, if viewed as a function of x, is also a DC-function, when for each class z, $x \mapsto h(x, z)$ is a DC-function. We then present a DCA solution for the adversarial attack computation for adversarial robustness.

For a real-valued hypothesis h, the multi-class margin $\rho_h(x, y)$ for a labeled pair (x, y) is defined by

$$\rho_h(x, y) = h(x, y) - \max_{y' \neq y} h(x, y'). \tag{5}$$

Table 1 shows the standard and adversarial loss, where $\Phi \colon \mathbb{R} \to \mathbb{R}_+$ is a non-increasing function upper bounding the indicator function $t \mapsto \mathbb{1}_{t \leq 0}$. Here, $\| * \| \cdot$ denotes a norm on the input feature space $\mathcal{X}$, and γ is the size of a perturbation. Observe that since Φ is non-increasing, we can rewrite the adversarial margin-based loss as the following equality [60]:

$$\widetilde{\Phi}(h, x, y) = \sup_{x' \colon \|x - x'\| \leq \gamma} \Phi(\rho_h(x', y)) = \Phi\left(\inf_{x' \colon \|x - x'\| \leq \gamma} \rho_h(x', y)\right). \tag{6}$$

Table 1 Target loss and surrogate margin-based loss in standard and adversarial classification

	Standard classification	Adversarial classification
Target loss	$\ell_{0-1} = \mathbb{1}_{\rho_h(x,y)\leq 0}$	$\ell_\gamma = \sup_{x' \colon \|x-x'\|\leq\gamma} \mathbb{1}_{\rho_h(x',y)\leq 0}$
Surrogate margin-based loss	$\Phi(\rho_h(x, y))$	$\widetilde{\Phi} = \sup_{x' \colon \|x-x'\|\leq\gamma} \Phi(\rho_h(x', y))$

Thus, to optimize $\widetilde{\Phi}$, we need to solve the following adversarial attack computation problem for each labeled pair (x, y), often with the hypothesis h being a neural network:

$$\min_{x' \colon \|x-x'\|\leq\gamma} \rho_h(x', y) = \min_{x' \colon \|x-x'\|\leq\gamma} \left(h(x', y) - \max_{y'\neq y} h(x', y') \right). \tag{7}$$

We now show that for a fixed y, the function $x \mapsto \rho_h(x, y)$ admits a DC-decomposition when for each class z, the function $x \mapsto h(x, z)$ is a DC-function. This condition is satisfied by all neural networks commonly used in practice (see Sect. 3).

Theorem 6 *Assume that for each class $z \in \{1, \ldots, c\}$, $h(x, z)$ admits the DC-decomposition $h(x, z) = f_z(x) - g_z(x)$, where f_z and g_z are convex functions. Then, for any y, the function $x \mapsto \rho_h(x, y)$ admits the following DC-decomposition:*

$$\rho_h(x, y) = \left(f_y(x) + \sum_{z\in\mathcal{Y} \colon z\neq y} g_z(x) \right) - \max_{y'\neq y} \left(f_{y'}(x) + \sum_{z\in\mathcal{Y} \colon z\neq y'} g_z(x) \right),$$

where $x \mapsto (f_y(x) + \sum_{z\in\mathcal{Y} \colon z\neq y} g_z(x))$ and $x \mapsto \max_{y'\neq y}(f_{y'}(x) + \sum_{z\in\mathcal{Y} \colon z\neq y'} g_z(x))$ are convex functions with respect to x.

Proof For a labeled example (x, y), the margin can be expressed as follows:

$$\begin{aligned}
\rho_h(x, y) &= h(x, y) - \max_{y'\neq y} h(x, y') \\
&= \big(f_y(x) - g_y(x)\big) - \max_{y'\neq y} \big(f_{y'}(x) - g_{y'}(x)\big) \\
&= \left(f_y(x) + \sum_{z\in\mathcal{Y}} g_z(x) - g_y(x) \right) - \max_{y'\neq y} \left(f_{y'}(x) + \sum_{z\in\mathcal{Y}} g_z(x) - g_{y'}(x) \right) \\
&= \left(f_y(x) + \sum_{z\in\mathcal{Y} \colon z\neq y} g_z(x) \right) - \max_{y'\neq y} \left(f_{y'}(x) + \sum_{z\in\mathcal{Y} \colon z\neq y'} g_z(x) \right).
\end{aligned}$$

The convexity of the function $x \mapsto (f_y(x) + \sum_{z\in\mathcal{Y} \colon z\neq y} g_z(x))$ and that of $x \mapsto \max_{y'\neq y}(f_{y'}(x) + \sum_{z\in\mathcal{Y} \colon z\neq y'} g_z(x))$ hold by the assumption and the fact that convexity is preserved under sum and pointwise maximum. ◻

As discussed in Sect. 1, the computation of the adversarial attack for adversarial robustness is an important problem in practice, with the form in (7), for a given hypothesis $h \in \mathcal{H}$, sample (x, y) and perturbation size γ. Combining the results of Sect. 3 and Theorem 6, the problem can be cast as an instance of DC-programming [46, 47], for which we can make use of the DCA algorithm. The pseudocode of our algorithm to solve the optimization problem (7) is given in Algorithm 3. This provides a DCA-based solution for the computation of the

Algorithm 3 DCA solution for the computation of the adversarial attack for adversarial robustness.

```
input: Maximum number of iterations $T$, hypothesis $h_{\mathrm{NN-adv}} \in \mathcal{H}_{\mathrm{NN-adv}}$, example $(x, y)$, perturbation size $\gamma$, initial point $x'_1$, tolerances $\epsilon_1$ and $\epsilon_2$
decompose by Section 3: $h_{\mathrm{NN-adv}}(x, z) = f_z(x) - g_z(x)$, for $z \in \{*\}1, \ldots, c$
decompose by Theorem 6: $\rho_{h_{\mathrm{NN-adv}}}(x, y) = f_{\mathrm{NN-adv}}(x) - g_{\mathrm{NN-adv}}(x)$, where
    $$f_{\mathrm{NN-adv}}(x) = f_y(x) + \sum_{z \in \mathcal{Y}:\, z \neq y} g_z(x), \quad g_{\mathrm{NN-adv}}(x) = \max_{y' \neq y}(*) f_{y'}(x) + \sum_{z \in \mathcal{Y}:\, z \neq y'} g_z(x)$$
for each round $t = 1, \ldots, T$ do
    $x'_{t+1} \in \operatorname{argmin}\{*\} f_{\mathrm{NN-adv}}(x') - g_{\mathrm{NN-adv}}(x'_t) - \partial g_{\mathrm{NN-adv}}(x'_t) \cdot (x' - x'_t) \colon \|x - x'\| \le \gamma$
    if $|(f_{\mathrm{NN-adv}} - g_{\mathrm{NN-adv}})(x'_{t+1}) - (f_{\mathrm{NN-adv}} - g_{\mathrm{NN-adv}})(x'_t)| \le \epsilon_1$ or $\| * \|x'_{t+1} - x'_t \le \epsilon_2$ then
        break
    end if
end for
```

adversarial attack for the family of feedforward neural networks and that of convolutional neural networks, denoted as $\mathcal{H}_{\mathrm{NN-adv}}$.

Here, we wish to further emphasize the significance of our DCA solution for the computation of the adversarial attack for adversarial robustness. The adversarial attack computation problem has been extensively studied in the adversarial robustness literature, which is crucial for both adversarial training and evaluation. The existing typical methods for the computation of the adversarial attack include the single-step *Fast Gradient Sign Method (FGSM)* [15], its stronger version *Projected Gradient Descent (PGD)* method [25, 37], and the state-of-the-art *Auto-PGD (APGD)* method [12]. To compare with our Algorithm 3, we describe FGSM and PGD for solving (7) with ℓ_∞ norm in the following, and refer interested readers to [12, Algorithm 1] for details of APGD. Here, we adopt the same notation as for Algorithm 3.

- Fast Gradient Sign Method (FGSM):

$$x' = x - \gamma \operatorname{sign}(\partial \rho_{h_{\mathrm{NN-adv}}}(x, y)).$$

- Projected Gradient Descent (PGD): for each round $t = 1, \ldots, T$,

$$x'_{t+1} = \mathrm{Proj}_{\{x' : \|x - x'\| \le \gamma\}}(x'_t - \alpha \cdot \operatorname{sign}(\partial \rho_{h_{\mathrm{NN-adv}}}(x'_t, y))),$$

  where α is the step size and $\mathrm{Proj}_B(\beta) := \operatorname{argmin}_{\beta' \in B} \|\beta - \beta'\|_2$ is the projection.

Nevertheless, none of these methods is guaranteed to produce the strongest attack, that is, the margin minimizer. On the one hand, it has been observed that using a single-step method like FGSM during adversarial training suffers from the issue of "catastrophic overfitting" [58], which suggests that stronger attack method or real margin minimization is needed during the training. On the other hand, [12, Table 2] show that the adversarial test accuracy for most of the studied defenses can be further reduced if evaluated on APGD, which implies that previous methods are not really reliable for the adversarial evaluation. Instead, our solution for the adversarial attack computation is built upon the DCA algorithm and benefits from its convergence guarantees [46], which can produce stronger attacks than PGD and is also comparable with the state-of-the-art method APGD in certain cases as shown in Sect. 5. Furthermore, our proposed method can also be combined or incorporated into other targeted attack or ensemble attack methods like AutoAttack [12], which can be helpful for both adversarial training and evaluation.

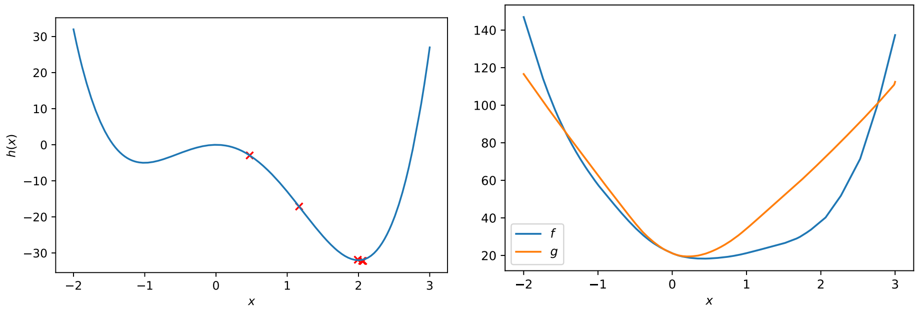

Fig. 2 Left: a fourth degree polynomial $F: x \mapsto 3x^4 - 4x^3 - 12x^2$. Right: DC-components $f_{\mathrm{NN-aprox}}$ and $g_{\mathrm{NN-aprox}}$ of the trained two-hidden-layer neural network

5 Experiments

Here, we present experiments on simulated data and real-world datasets to validate the effectiveness of our algorithms for the approximate optimization and adversarial attack computation problems. In the approximate optimization example, we show that our DCA solution, Algorithm 2, can effectively minimize a polynomial function with only access to its samples using a feedforward neural network. In the adversarial attack computation example, we show that our DCA solution, Algorithm 3, can effectively achieve margin minimization for an adversarially trained convolutional neural network on CIFAR-10 [23].

Approximate optimization. The function to minimize is a fourth-degree polynomial $F: x \mapsto 3x^4 - 4x^3 - 12x^2$ whose plot is shown in Fig. 2. Its derivative is $F'(x) = 12(x+1)x(x-2)$. Function F admits a global minimum $F(2) = -32$ at $x = 2$ and a local minimum $F(-1) = -5$ at $x = -1$. Assume that we have access to samples $\{(x^i, F(x^i))\}_{i=1}^m$, where $\{x^i\}_{i=1}^m$ are drawn from the uniform distribution on the interval $[-2, 3]$. We train a feedforward neural network with 2 hidden-layers and 100 units in each layer using 5,000 training samples. As shown in Theorem 2, the trained neural network, denoted by $h_{\mathrm{NN-aprox}}$, admits a DC-decomposition $h_{\mathrm{NN-aprox}} = f_{\mathrm{NN-aprox}} - g_{\mathrm{NN-aprox}}$. The two convex DC-components $f_{\mathrm{NN-aprox}}$ and $g_{\mathrm{NN-aprox}}$ are illustrated in Fig. 2. With a randomly chosen initial point $x = 0.47$, DCA rapidly converges to $h_{\mathrm{NN-aprox}}(2.05) = -32.20$. The iteration points are marked in Fig. 2, which closely match the polynomial F and approach its minimum value, though the form of F is unknown to the neural network.

Adversarial attacks for adversarial robustness. Here, we also report the computation results of the adversarial attack for an adversarially trained convolutional neural network on images from CIFAR-10 [23]. The architecture of the CNN is described in Table 2. The adversarial attacks are ℓ_∞-norm bounded perturbations of size $\gamma = 8/255$. Figure 3 displays histograms of the margin value over 100 trials for each image. We ran 10-step PGD on the margin loss with random starts and then ran DCA for 20 rounds starting from the returned point. The vertical line indicates the margin value corresponding to the adversarial attack found by 100-step APGD on the margin loss. Figure 3 shows that DCA can improve upon PGD attacks and is comparable with strong APGD attacks.

Table 2 Architecture of the convolutional neural network used in Sect. 5 for the computation of the adversarial attack

Layer	Kernel (Weight) size	Stride
Convolution	$3 \times 3 \times 3 \times 32$	1×1
ReLU	–	–
Average pooling	2×2	2×2
Convolution	$3 \times 3 \times 32 \times 64$	1×1
ReLU	–	–
Average pooling	2×2	2×2
Convolution	$3 \times 3 \times 64 \times 64$	1×1
ReLU	–	–
Fully connected	64×1024	–
ReLU	–	–
Fully connected	10×64	–

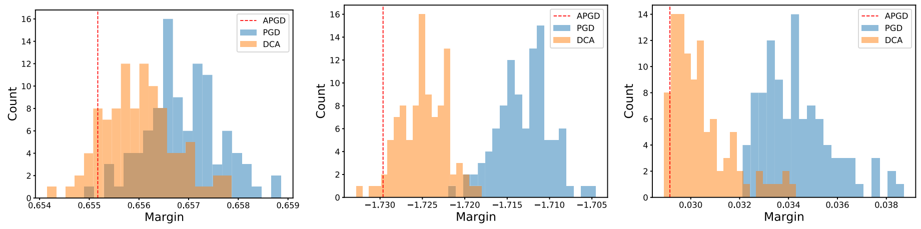

Fig. 3 Histograms of the margin value over 100 trials for each image. We ran 10-step PGD on the margin loss with random starts and then ran DCA for 20 rounds starting from the returned point. The vertical line indicates the margin value corresponding to the adversarial attack found by 100-step APGD on the margin loss

6 Conclusion

We presented the study of two key problems related to neural network optimization: the computation of the adversarial attack for adversarial robustness and approximate optimization based on neural networks. Our results include new DCA solutions which are shown to be effective, as demonstrated by our experiments. Our results can help design better adversarial training algorithms or stronger adversarial attacks during evaluation in adversarial robustness. They can be extended to many other neural network architectures using a similar proof technique, and can be extended to many other neural network learning and optimization problems.

Acknowledgements We thank our colleague Satyen Kale, as well as the editor and reviewers, for helpful comments on an earlier draft of this manuscript.

Data availibility The datasets analysed during the current study are available in the CIFAR-10 repository, https://www.cs.toronto.edu/\protect\unhbox\voidb@x\penalty\@M\kriz/cifar.html.

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Argyriou, A., Hauser, R., Micchelli, C.A., Pontil, M.: A DC-programming algorithm for kernel selection. In: International Conference on Machine Learning, pp. 41–48 (2006)
2. Awasthi, P., Frank, N., Mao, A., Mohri, M., Zhong, Y.: Calibration and consistency of adversarial surrogate losses. In: Advances in Neural Information Processing Systems, pp. 9804–9815 (2021)
3. Awasthi, P., Mao, A., Mohri, M., Zhong, Y.: A finer calibration analysis for adversarial robustness. arXiv preprint arXiv:2105.01550 (2021)
4. Awasthi, P., Mao, A., Mohri, M., Zhong, Y.: H-consistency bounds for surrogate loss minimizers. In: International Conference on Machine Learning, pp. 1117–1174 (2022)
5. Awasthi, P., Mao, A., Mohri, M., Zhong, Y.: Multi-class H-consistency bounds. In: Advances in Neural Information Processing Systems, pp. 782–795 (2022)
6. Awasthi, P., Cortes, C., Mohri, M.: Best-effort adaptation. https://doi.org/10.48550/arXiv.2305.05816 (2023)
7. Awasthi, P., Mao, A., Mohri, M., Zhong, Y.: Theoretically grounded loss functions and algorithms for adversarial robustness. In: International Conference on Artificial Intelligence and Statistics, pp. 10077–10094 (2023)
8. Bartlett, P.L., Jordan, M.I., McAuliffe, J.D.: Convexity, classification, and risk bounds. J. Am. Stat. Assoc. **101**(473), 138–156 (2006)
9. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: IEEE Symposium on Security and Privacy (SP), pp. 39–57 (2017)
10. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
11. Cortes, C., Mohri, M., Suresh, A.T., Zhang, N.: A discriminative technique for multiple-source adaptation. In: International Conference on Machine Learning, pp. 2132–2143 (2021)
12. Croce, F., Hein, M.: Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: International Conference on Machine Learning, pp. 2206–2216 (2020)
13. Wenlong, F., Tingsong, D., Zhai, J.: A new branch and bound algorithm based on DC-composition about nonconvex quadratic programming with box constrained. J. Math. Study **46**(3), 311–318 (2013)
14. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
15. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
16. Hiroshi, K., Thien, T.P., Hoang, T.: Optimization on Low Rank Nonconvex Structures, Volume 15. Springer (2013)
17. Hoang, T.: Concave programming under linear constraints. Transl. Sov. Math. **5**, 1437–1440 (1964)
18. Hoang, T.: Counter-examples to some results on D.C. optimization. Technical report, Institute of Mathematics, Hanoi, Vietnam (2002)
19. Hoffman, J., Mohri, M., Zhang, N.: Algorithms and theory for multiple-source adaptation. In: Advances in Neural Information Processing Systems, pp. 8256–8266 (2018)
20. Hoffman, J., Mohri, M., Zhang, N.: Multiple-source adaptation theory and algorithms. Ann. Math. Artif. Intell. **89**(3–4), 237–270 (2021)
21. Hoffman, J., Mohri, M., Zhang, N.: Multiple-source adaptation theory and algorithms - addendum. Ann. Math. Artif. Intell. **90**(6), 569–572 (2022)
22. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5), 359–366 (1989)
23. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Technical report (2009)
24. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
25. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. International Conference on Learning Representations workshop (2016)
26. Kuznetsov, V., Mohri, M.: Learning theory and algorithms for forecasting non-stationary time series. In: Advances in Neural Information Processing Systems, pp. 541–549 (2015)

27. Kuznetsov, V., Mohri, M.: Time series prediction and online learning. In: Conference on Learning Theory, pp. 1190–1213 (2016)
28. Kuznetsov, V., Mohri, M.: Discrepancy-based theory and algorithms for forecasting non-stationary time series. Ann. Math. Artif. Intell. **88**(4), 367–399 (2020)
29. Kuznetsov, V., Mohri, M., Syed, U.: Multi-class deep boosting. In: Advances in Neural Information Processing Systems, pp. 2501–2509 (2014)
30. Le Thi, H.A., Dinh, T.P.: A branch and bound method via dc optimization algorithms and ellipsoidal technique for box constrained nonconvex quadratic problems. J. Glob. Optim. **13**(2), 171–206 (1998)
31. Le Thi, H.A., Dinh, T.P.: The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. Ann. Oper. Research **133**(1), 23–46 (2005)
32. Le Thi, H. A., Pham Dinh, T.: DC Programming and DCA for Nonconvex Optimization: Theory, Algorithms and Applications. MAMERN09 (2009)
33. Le Thi, H.A., Dinh, T.P.: Recent advances in DC programming and DCA. Trans. Comput. Intell. 13:1–37 (2014)
34. Le Thi, H.A., Dinh, T.P.: DC programming and DCA: thirty years of developments. Math. Program. **169**(1), 5–68 (2018)
35. Le Thi, H.A., Dinh, T.P., Dung, M.L.: A combined DC optimization-ellipsoidal branch-and-bound algorithm for solving nonconvex quadratic programming problems. J. Combin. Optim. **2**(1), 9–28 (1998)
36. Long, P., Servedio, R.: Consistency versus realizable H-consistency for multiclass classification. In: International Conference on Machine Learning, pp. 801–809 (2013)
37. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
38. Mao, A., Mohri, C., Mohri, M., Zhong, Y.: Two-stage learning to defer with multiple experts. In: Advances in Neural Information Processing Systems (2023)
39. Mao, A., Mohri, M., Zhong, Y.: H-consistency bounds: Characterization and extensions. In Advances in Neural Information Processing Systems (2023)
40. Mao, A., Mohri, M., Zhong, Y.: Theoretical analysis and applications. In: Cross-Entropy Loss Functions: International Conference on Machine Learning (2023)
41. Mao, A., Mohri, M., Zhong, Y.: Structured prediction with stronger consistency guarantees. In: Advances in Neural Information Processing Systems (2023)
42. Mao, A., Mohri, M., Zhong, Y.: H-consistency bounds for pairwise misranking loss surrogates. In: International Conference on Machine Learning (2023)
43. Mohri, M., Muñoz Medina, A.: Learning algorithms for second-price auctions with reserve. J. Mach. Learn. Res. **17**(1), 2632–2656 (2016)
44. Mohri, M., Medina, A.M.: Learning theory and algorithms for revenue optimization in second price auctions with reserve. In: International Conference on Machine Learning, pp. 262–270 (2014)
45. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning, 2nd Ed. MIT Press (2018)
46. Pham Dinh, T., Le Thi, H.A.: Convex analysis approach to DC programming: theory, algorithms and applications. Acta mathematica vietnamica **22**(1), 289–355 (1997)
47. Pham Dinh, T., Le Thi, H.A: A DC optimization algorithm for solving the trust-region subproblem. SIAM J. Optim. **8**(2), 476–505 (1998)
48. Reiner, H., Hoang, T.: Global Optimization: Deterministic Approaches. Springer, Berlin (2013)
49. Reiner, H., Thoai, N.V.: DC Programming: Overview. J. Optim. Theory Appl. **103**(1), 1–43 (1999)
50. Seck, I., Loosli, G., Canu, S., Niu, Y.-S.: Difference-of-convex algorithm applied to adversarial robustness verification (2019). https://ailab.criteo.com/wp-content/uploads/2019/10/9.-Seck.pdf
51. Sriperumbudur, B.K., Lanckriet, G.R.G.: A proof of convergence of the concave-convex procedure using Zangwill's theory. Neural Comput. **24**(6), 1391–1407 (2012)
52. Sriperumbudur, B.K., Torres, D.A., Lanckriet, G.R.G.: Sparse Eigen Methods by D.C. Programming. In: International Conference on Machine Learning, pp. 831–838 (2007)
53. Steinwart, I.: How to compare different loss functions and their risks. Construct. Approxim. **26**(2), 225–287 (2007)
54. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)
55. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
56. Tewari, A., Bartlett, P.L.: On the consistency of multiclass classification methods. J. Mach. Learn. Res. **8**(36), 1007–1025 (2007)
57. Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., Madry, A.: Robustness may be at odds with accuracy. arXiv preprint arXiv:1805.12152 (2018)

58. Wong, E., Schmidt, F., Metzen, J.H., Zico Kolter, J.: Scaling provable adversarial defenses. Advances in Neural Information Processing Systems (2018)
59. Yen, I.E.H., Peng, N., Wang, P.-O., Lin, S.-D.: On convergence rate of concave-convex procedure. In Proceedings of the NIPS 2012 Optimization Workshop, pp. 31–35 (2012)
60. Yin, D., Ramchandran, K., Bartlett, P.L.: Rademacher complexity for adversarially robust generalization. In: International Conference of Machine Learning, pp. 7085–7094 (2019)
61. Yuille, A.L., Rangarajan, A.: The concave-convex procedure. Neural Comput. **15**(4), 915–936 (2003)
62. Zhang, M., Agarwal, S.: Bayes consistency vs. H-consistency: the interplay between surrogate loss functions and the scoring function class. In: Advances in Neural Information Processing Systems, pp. 16927–16936 (2020)
63. Zhang, T.: Statistical behavior and consistency of classification methods based on convex risk minimization. Ann. Stat. **32**(1), 56–85 (2004)