ISSN 1064-5624, Doklady Mathematics, 2023, Vol. 108, Suppl. 2, pp. S233–S241. © Pleiades Publishing, Ltd., 2023.
ISSN 1064-5624, Doklady Mathematics, 2023. © Pleiades Publishing, Ltd., 2023.
Russian Text © The Author(s), 2023, published in Doklady Rossiiskoi Akademii Nauk. Matematika, Informatika, Protsessy Upravleniya, 2023, Vol. 514, No. 2, pp. 80–90.

A New Computationally Simple Approach for Implementing Neural Networks with Output Hard Constraints

A. V. Konstantinov[a],* and L. V. Utkin[a],**

Received August 9, 2023; revised September 25, 2023; accepted October 15, 2023

Abstract—A new computationally simple method of imposing hard convex constraints on the neural network output values is proposed. The key idea is to map a latent vector to a point that is guaranteed to be inside the feasible set defined by a set of constraints. The mapping is implemented by the additional neural network layer with constraints for output. The method proposed is extended to the case when constraints are imposed not only on the output vectors, but also on joint constraints depending on inputs. The projection approach to imposing constraints on outputs can simply be implemented in the framework of the proposed method. It is shown how to incorporate different types of constraints into the proposed method, including linear and quadratic constraints, equality constraints, dynamic constraints, and constraints in the form of boundaries. An important feature of the method is its computational simplicity. Complexities of the forward pass of the proposed neural network layer by linear and quadratic constraints are $O(nm)$ and $O(n^2m)$, respectively, where n is the number of variables and m is the number of constraints. Numerical experiments illustrate the method by solving optimization and classification problems. The code implementing the method is publicly available.

Keywords: neural network, hard constraints, convex set, projection model, optimization problem, classification

DOI: 10.1134/S1064562423701077

1. INTRODUCTION

Neural networks can be regarded as an important and effective tool for solving various machine learning problems. In a number of applied tasks, the implementation of which is carried out using neural networks, it is required that the network output be limited, i.e., the predictions meet specified constraints. Examples of problems that require constraints on network output values are constrained optimization problems, models generating images or parts of images in a given region, classification problems with constraints on class probabilities, etc. The most common approach to account for such constraints is to add some additional penalty terms to the loss function to penalize constraint violations. However, this does not guarantee that the constraints will be satisfied for all training and new examples, since in this case, violation of the constraints is only penalized, but not eliminated. Therefore, this approach leads to the so-called *soft* constraints [1]. Another approach is to modify the neural network so that it predicts strictly within a limited output space. In this case the constraints are *hard* in the sense that they are satisfied for any input example during training and use [2].

Although many applications require hard constraints, there are currently not many models implementing them. Most models are based on the use of soft constraints due to their simple implementation [3]. A method taking into account rigid conical constraints of the form $Ax \le 0$ was proposed in [2]. The corresponding model puts predictions into a feasible region using a specific set of rays. A serious limitation of this method is the need to search for corresponding rays. If this approach is applied not only to conic constraints, it is necessary to look for all the vertices of the set. However, the number of vertices can be extremely large. A general approach to solving optimization problems with constraints is presented in [4]. It aims to incorporate (potentially nonconvex) equality and inequality constraints into deep learning-based optimization algorithms. Its performance largely depends on the training process and the chosen model architecture. A neural network architecture with hard constraints on output data using an additional output layer is proposed in [5]. However, the method considers only linear constraints, but the main thing is that its implementation requires knowledge of all vertices of

[a] *Higher School of Artificial Intelligence Technologies, Peter the Great St. Petersburg Polytechnic University, St. Petersburg, 195251 Russia*

**e-mail: andrue.konst@gmail.com*

***e-mail: lev.utkin@gmail.com*

the constraint polyhedron, the number of which can be huge.

An interesting approach to solving quadratic optimization problems with linear constraints was proposed in [6]. A similar approach, which embeds an optimization layer into a neural network, is proposed in [7]. Unlike [6], this approach is extended to the case of an arbitrary convex loss function, including its parameters. According to [7], the operation of projecting an output point onto a bounded domain can be implemented using a differentiable optimization layer, which ensures that the output of the neural network satisfies the constraints. However, these approaches require solving convex optimization problems for each forward pass of the neural network, which significantly complicates their implementation. Another method for solving an optimization problem with linear constraints is presented in [8]. It should be noted that the method may require significant computational resources and time to solve complex optimization problems. Moreover, it solves optimization problems with linear constraints only. Several approaches to solving constrained optimization problems were proposed in [9–11]. An analysis of these approaches can be found in review articles [12, 13].

As far as we know, there is currently no approach that allows us to implement and train neural networks whose output satisfies linear and quadratic constraints without solving the optimization problem with a forward pass of the neural network. Therefore, we present a new computationally simple method that imposes strong linear and quadratic constraints on the output values of a neural network. The main idea of the method is to map the vector of latent parameters of the neural network to a point that is guaranteed to be inside the admissible set defined by a set of constraints. The mapping is implemented by a special additional layer of the neural network. The method proposed can be easily generalized to the case where constraints are imposed not only on the output data of the network, but also on the input data. Another feature of the approach is that the method of projecting a point onto an admissible set is simply implemented within the framework of this approach. An important feature of the proposed method is its computational simplicity. For example, the computational complexity of the direct pass of the neural network layer implementing the method in the case of linear constraints is $O(nm)$, and in the case of quadratic constraints $O(n^2m)$, where n us number of variables and m is the number of constraints.

The method proposed can be used in various applied problems. First of all, this is the solution of optimization problems with arbitrary differentiable loss functions and with linear or quadratic constraints. The method can be used to implement generative models with constraints. It can be used when constraints are imposed on specific subsets of points. There are many other applications where the input and output of neural networks must be constrained.

The contribution of the work can be summarized as follows:

1. A new computationally simple method is proposed that imposes hard linear and quadratic constraints on the output values of a neural network.

2. The implementation of the method with various types of constraints is considered, including linear and quadratic constraints, equality constraints, and constraints imposed on input and output data.

3. Various modifications of the proposed method are studied, including a model for obtaining solutions on the boundaries of the feasible set and projection models.

4. Numerical experiments illustrating the proposed method are presented. In particular, various optimization problems and classification problems are considered.

The software that implements the method can be found on the website: https://github.com/andruekonst/ConstraiNet/.

2. PROBLEM STATEMENT AND PROPOSED METHOD

Let $z \in \mathbb{R}^d$ be the input vector of a neural network and $x \in \mathbb{R}^n$ be the output vector (prediction) of this network. The neural network is considered a function $f_\theta : \mathbb{R}^d \to \mathbb{R}^n$ such that $x = f_\theta(z)$, where $\theta \in \Theta$ is the vector of trained parameters.

Let $\Omega \subset \mathbb{R}^n$ be a convex feasible set of output data formed by a set of constraints in the form of m inequalities:

$$\Omega = \{x \,|\, h_i(x) \le 0,\ i = 1,\ldots,m\}, \tag{1}$$

where each constraint $h_i(x) \le 0$ is convex.

Our goal is to build a neural network with constraints on the output data. In other words, it is necessary to build a model $x = f_\theta(z) : \mathbb{R}^d \to \Omega$ and impose constraints on x so that $x \in \Omega$ for any $z \in \mathbb{R}^d$, i.e.,

$$\forall z \in \mathbb{R}^d \,(f_\theta(z) \in \Omega).$$

2.1. Neural Network Layer with Constraints on Output Data

Given a fixed point p inside the convex domain Ω, i.e., $p \in \Omega$. Any point x from many Ω can be represented as

$$x = p + \alpha \cdot r,$$

where $\alpha \ge 0$ is a scale parameter and $r \in \mathbb{R}^n$ is a vector (ray from point p).

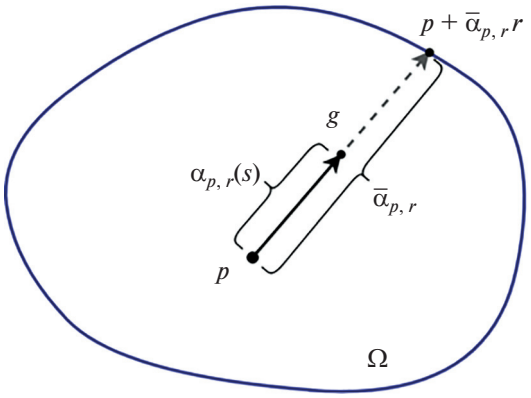

Fig. 1. Mapping $g_p(r,s)$.

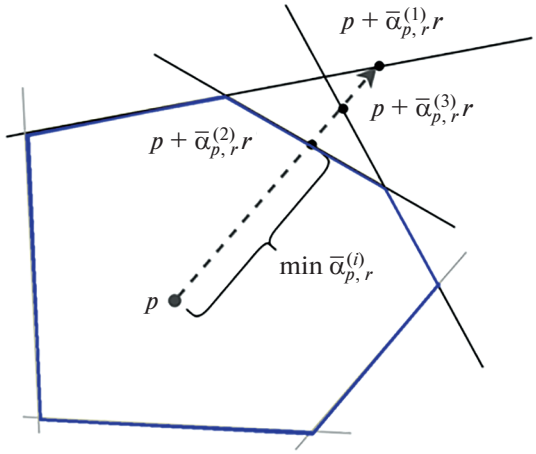

Fig. 2. Illustration of the search for the upper bound $\bar{\alpha}_{p,r}$ of crossing constraints.

On the other hand, for any p and r, there is an upper bound $\bar{\alpha}_{p,r}$ of the parameter α, defined as

$$\bar{\alpha}_{p,r} = \max\{\alpha \geq 0 \,|\, p + \alpha \cdot r \in \Omega\}.$$

Moreover, the segment $[p; p + \bar{\alpha}_{p,r} \cdot r]$ belongs to Ω, because Ω is a convex set. The meaning of determining the boundary $\bar{\alpha}_{p,r}$ is to determine the point of intersection of the ray r with one of the constraints.

We define a neural network layer that maps the ray r and scale s as

$$g_p(r,s) = p + \alpha_{p,r}(s) \cdot r,$$

where $\alpha_{p,r}(s)$ is a function of the layer parameter s and $\bar{\alpha}_{p,r}$, defined as

$$\alpha_{p,r}(s) = \sigma(s) \cdot \bar{\alpha}_{p,r},$$

$\sigma(s) : \mathbb{R} \to [0,1]$ is a sigmoid function, i.e., a smooth monotonic function.

Such a layer is guaranteed to satisfy the following constraints:

$$\forall r \in \mathbb{R}^n, \quad s \in \mathbb{R}(g_p(r,s) \in \Omega),$$

since the segment $[p, p + \bar{\alpha}_{p,r} \cdot r]$ belongs Ω.

The mapping of the ray r and scalar s to a point inside the domain Ω is schematically shown in Fig. 1. From the ray r issuing from the point p, the intersection with the domain's boundary $p + \bar{\alpha}_{p,r} \cdot r$ is found and, as a result of scaling, the point g is obtained.

For a system of constraints, it suffices to find the upper bound $\bar{\alpha}_{p,r}$ satisfying each constraint. Let $\bar{\alpha}^{(i)}_{p,r}$ be the upper bound of the parameter α, corresponding to the ith constraint $(h_i(x) \leq 0)$ of system (1). Then, the upper bound of the entire system is specified so that $[p, p + \bar{\alpha}_{p,r} \cdot r] \subseteq [p, p + \bar{\alpha}^{(i)}_{p,r} \cdot r]$, i.e.,

$$\bar{\alpha}_{p,r} = \min\{\bar{\alpha}^{(i)}_{p,r}\}_{i=1}^{m}. \tag{2}$$

The search for the upper bound $\bar{\alpha}_{p,r}$ under linear constraints is schematically shown in Fig. 2.

The computational complexity of the forward pass of the described neural network layer is directly proportional to the number of constraints and the computational complexity of finding the intersection with one constraint.

Proposition 1. *Any vector $x \in \Omega$ can be represented using a layer $g_p(r,s)$. For any input (r,s), the output of the layer $g_p(r,s)$ belongs to the set Ω.*

Proof:

1. Any output vector $g_p(r,s)$ satisfies the constraints; i.e., $\forall r \in \mathbb{R}^n$, $s \in \mathbb{R}$, the condition $g_p(r,s) \in \Omega$ is satisfied, because $\alpha_{p,r}(s) \leq \bar{\alpha}^{(i)}_{p,r}$ and $[p,\ p + \bar{\alpha}^{(i)}_{p,r} \cdot r] \subset \Omega$. Hence, $g_p(r,s) \in [p, p + \alpha_{p,r}(s)] \subset \Omega$.

2. Any point $x \in \Omega$ can be represented using a layer $g_p(r,s)$. Indeed, let $r = x - p$, $s \to +\infty$. Then, $x = g_p(x - p, +\infty) = p + 1 \cdot (x - p) = x$, which was to be proved.

To obtain the model $f_\theta(z) : \mathbb{R}^d \to \Omega$, it is required to apply to the input of the layer $g_p(r,s)$ the output of the main neural network, $r_\theta(z)$ and $s_\theta(z)$:

$$f_\theta(z) = g(r_\theta(z), s_\theta(z)).$$

This aggregate model also forms a neural network, which can be trained by a backpropagation algorithm.

2.2. Linear Constraints

In the case of linear constraints, $\bar{\alpha}_{p,r}$ is determined by the intersection of a ray from a point p in the direction r with the set of constraints. Consider an intersection with one linear constraint of the form $a_i^{\mathrm{T}} x \leq b_i$.

The upper bound of the parameter α is determined by the solution of the system

$$\begin{cases} x = p + \alpha \cdot r \\ a_i^{\mathrm{T}} x = b_i \\ \alpha \geq 0. \end{cases} \tag{3}$$

Therefore, if a solution exists, then the following is true:

$$a_i^{\mathrm{T}} p + \alpha \cdot a_i^{\mathrm{T}} r = b_i,$$

$$\overline{\alpha}_{p,r}^{(i)} = \frac{b_i - a_i^{\mathrm{T}} p}{a_i^{\mathrm{T}} r}.$$

If $a_i^{\mathrm{T}} r = 0$ or $\overline{\alpha}_i(p,r) < 0$, then (3) has no solution and $\overline{\alpha}_{p,r}^{(i)}$ can be set to $+\infty$. If a system of inequalities is given, then the upper bound is determined using (2).

In the case of linear constraints, the computational complexity of a direct pass through the neural network layer is $O(nm)$.

2.3. Quadratic Constraints

Let the ith quadratic constraint be given as

$$\frac{1}{2} x^{\mathrm{T}} P^{(i)} x + q_i^{\mathrm{T}} x \leq b_i,$$

where $P^{(i)}$ is a positive semidefinite matrix. Then, the intersection of the ray with the constraint is given by the equation

$$\frac{1}{2}(p + \alpha \cdot r)^{\mathrm{T}} P^{(i)} (p + \alpha \cdot r) + q_i^{\mathrm{T}} (p + \alpha \cdot r) = b_i.$$

Depending on the coefficient at α^2, two cases are possible:

1. If $r^{\mathrm{T}} P^{(i)} r = 0$, then the equation is linear and has a solution

$$\alpha = -\frac{q_i^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} P^{(i)} p - b_i}{p^{\mathrm{T}} P^{(i)} r + q_i^{\mathrm{T}} r}.$$

2. If $r^{\mathrm{T}} P^{(i)} r > 0$, then there are two solutions, but we consider only the larger positive one, corresponding to the movement in the direction of the ray. This solution is

$$\alpha = -\frac{-(p^{\mathrm{T}} P^{(i)} r + q_i^{\mathrm{T}} r) + \sqrt{D/4}}{r^{\mathrm{T}} P^{(i)} r},$$

$$\frac{D}{4} = (p^{\mathrm{T}} P^{(i)} r + q_i^{\mathrm{T}} r)^2 - (r^{\mathrm{T}} P^{(i)} r) \cdot (2 q_i^{\mathrm{T}} p + p^{\mathrm{T}} P^{(i)} p - 2 b_i),$$

since the denominator is positive.

The case of $r^{\mathrm{T}} P^{(i)} r < 0$ is impossible, since the matrix is positive semidefinite. Otherwise, the constraint would specify a nonconvex set.

If $\alpha \geq 0$, then $\overline{\alpha}_{p,r}^{(i)} = \alpha$. Otherwise, if the ray does not intersect the constraint, then $\overline{\alpha}_{p,r}^{(i)} = +\infty$. For a system of quadratic constraints, the upper bound has the form (2).

In the case of quadratic constraints, the computational complexity of the forward pass of the neural network layer is $O(n^2 m)$.

2.4. Equality Constraints

Consider the case when the domain Ω is specified by a system of linear equalities and inequalities of the form

$$x \in \Omega \Leftrightarrow \begin{cases} Ax \leq b \\ Qx = p. \end{cases} \tag{4}$$

In this case, the problem can be reduced to (1), i.e., to a system of inequalities. To do this, we find and fix a vector u satisfying the system $Qu = p$. If the system has no solutions, then Ω empty. If there is only one solution, then Ω consists of one point. Otherwise, there are an infinite number of solutions and it suffices to choose any of them, e.g., solving the least squares problem:

$$\|Qu - p\|^2 \to \min.$$

Then we can find the matrix R, which is a basis of the kernel Q; i.e., R satisfies the condition

$$\forall w (QRw = 0).$$

The matrix R can be obtained using the SVD decomposition of the matrix $Q \in \mathbb{R}^{\mu \times n}$:

$$USV = Q,$$

where $U \in \mathbb{R}^{\mu \times \mu}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $S \in \mathbb{R}^{\mu \times n}$ is a matrix with non-zero values only on the diagonal, sorted in descending order.

Then, the matrix R is given as

$$R = (v_1, \ldots, v_\delta),$$

where δ is the number of nonzero diagonal elements of the matrix S and $v_1, \ldots, v_\delta$ are the columns of the matrix V.

Hence,

$$\forall w \in \mathbb{R}^\delta (Q(Rw + u) = p).$$

The new system of constraints on the vector w is defined as

$$A(Rw + u) \leq b,$$

or, in canonical form,

$$Bw \le t, \tag{5}$$

where $B = AR$ and $t = b - Au$.

Thus, w is the vector of variables for the new system (4). For any w, a vector x satisfying the original system can be reconstructed in the form $x = Rw + u$. As a result, the model of the solution will be given as

$$f_\theta(z) = R\tilde{f}_\theta(z) + u, \tag{6}$$

where $\tilde{f}_\theta(z)$ is the model for constraints (5).

In a more general case, if an arbitrary convex set Ω is given as the intersection of convex inequality constraints (1) and one additional equality constraint:

$$x \in \Omega \Leftrightarrow \begin{cases} h_i(x) \le 0 \\ Qx = p, \end{cases}$$

then we can use a change of variables to obtain new (possibly nonlinear) constraints:

$$x \in \Omega \Leftrightarrow \begin{cases} h_i(Rw + u) \le 0 \\ x = Rw + u, \end{cases} \Leftrightarrow \begin{cases} \tilde{h}_i^{(R,u)}(w) \le 0 \\ x = Rw + u. \end{cases}$$

Thus, the model can be used to generate solutions $\tilde{f}_\theta(z)$ satisfying nonlinear constraints $\tilde{h}_i^{(R,u)}(\tilde{f}_\theta(z))$ and then solutions for x are obtained through (6).

2.5. Constraints on Input and Output Data

In practice, it may be needed not only to impose constraints on the output vector $f_\theta(z)$, but also to impose joint constraints depending on some input vectors. Let there be given a convex region of constraints on the input vector z and output vector $f_\theta(z)$:

$$\Lambda \subset \mathbb{R}^k \times \mathbb{R}^n,$$

i.e., for any z, the model $f_\theta(z)$ should satisfy the condition

$$y = \begin{bmatrix} f_\theta(z) \\ z \end{bmatrix} \in \Lambda.$$

Here, y is the concatenation of vectors $f_\theta(z)$ and z. If the domain is specified as the intersection of convex constraints,

$$\Lambda = \{y \mid \Gamma_i(y) \le 0,\ i = 1,\dots,m\},$$

then, for fixed z, we can construct by substitution a new system of constraints only on the output vector $f_\theta(z)$:

$$G(z) = \{z \mid \gamma_i(x;z) \le 0,\ i = 1,\dots,m\},$$

where $\gamma_i(x;z)$ are obtained by the substitution of z into Γ_i.

Here, γ_i depends on z as on fixed parameters and the only variable is x. For example, if Γ_i is a linear function, then, after substituting the parameters, $\gamma_i(x;z) \le 0$ will be a new linear constraint on x or be satisfied automatically. If Γ_i is a quadratic function, then the constraint on x will either be quadratic, linear, or satisfied automatically.

New *dynamic* constraints on output data depending on the input vector z satisfy the condition

$$f_\theta(z) \in G(z),$$

provided that the input vector z is from the feasible set:

$$z \in \left\{ z \mid \exists x : \begin{bmatrix} x \\ z \end{bmatrix} \in \Lambda \right\}.$$

Such constraints may change with z and can be implemented in a neural network provided that there is a fixed point p.

2.6. Projection Model

Note that using the proposed neural network layer with constraints on the output data, a model of a *projection* can be constructed that maps the points in Ω and has the property of idempotency, i.e.,

$$\forall x \in \mathbb{R}^n (f_\theta(f_\theta(x)) = f_\theta(x)).$$

This model can be implemented in two ways:

1. The first way is to train the model $f_\theta(z) = g(r_\theta(z), s_\theta(s))$ to identical transformation by minimizing a functional that penalizes the distance between the image and the preimage, e.g., for an orthogonal projection in the L_p –norm:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|f_\theta(x_i) - x_i\|_p. \tag{7}$$

This method can be used when it is necessary to build projection models for complex metrics, e.g., specified by neural networks.

2. The second way is the *central projection*, which can be obtained without optimizing the model, by specifying the ray $r_\theta(z) := z - p$. In this case, the scale must be specified without the sigmoid, explicitly, as $\alpha_{p,r}(s) = \min\{1, \bar{\alpha}_{p,r}\}$. Then,

$$\tilde{g}_p(r(x)) = p + \min\{1, \bar{\alpha}_{p,(x-p)}\} \cdot (x - p)$$
$$= \begin{cases} x, & \bar{\alpha}_{p,(x-p)} \ge 1 \\ (1 - \bar{\alpha}_{p,(x-p)})\, p + \bar{\alpha}_{p,(x-p)} x, & \text{otherwise.} \end{cases}$$

Examples of implementation of the projection model for quadratic constraints are shown in Fig. 3. For each of the options, a vector field is given, where the beginning of each arrow corresponds to the preimage and the end, to its projection into a set of 5 constraints. On the left is the result of a neural network whose parameters minimize (7), where Ω contains artifacts corresponding to regions with large approximation errors. On the right is the result of a neural net-

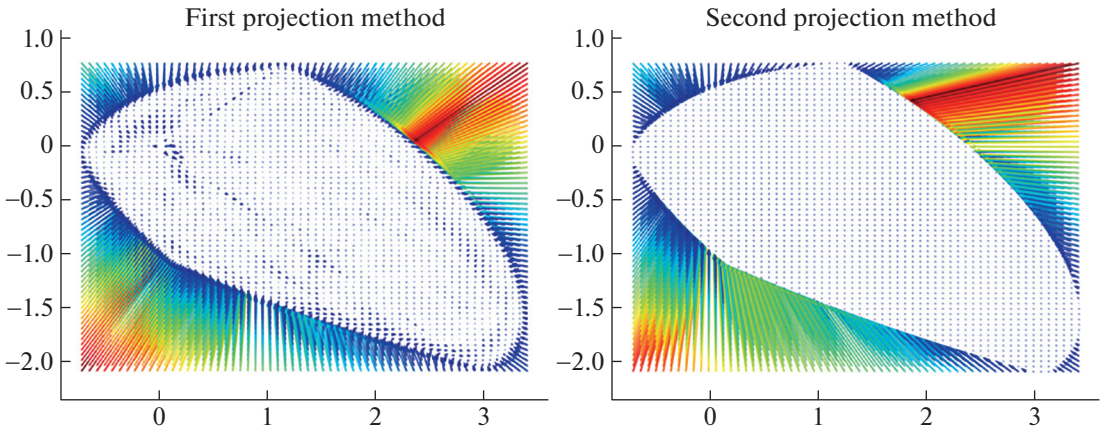

Fig. 3. Examples of projection models for quadratic constraints.

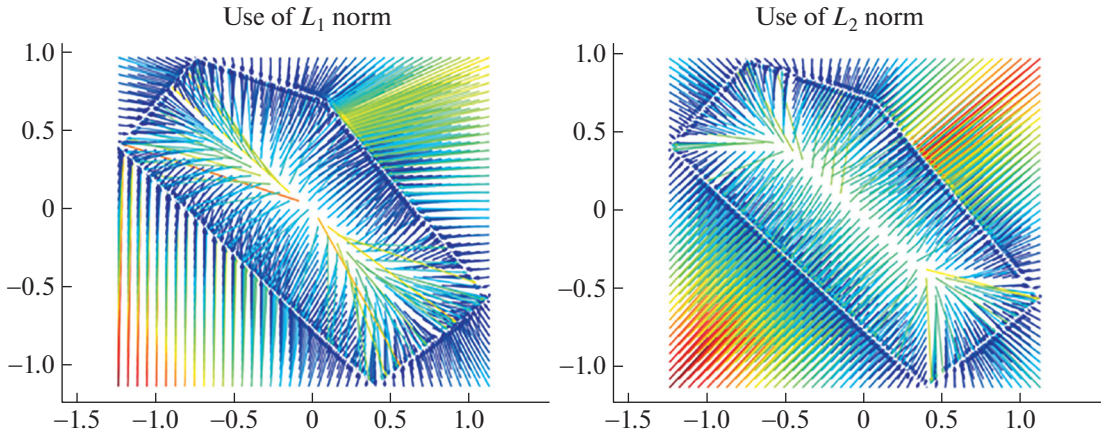

Fig. 4. Examples of approximation of projection onto a boundary using the L_1 and L_2 norms.

work without trained parameters, implementing a central projection where there are no errors.

2.7. Constructing Solutions at the Boundary

The method developed can also be applied to construct solutions in a nonconvex boundary set, denoted by $\partial\Omega$. To do this, we set $\sigma(s) = 1$. Then,

$$g(r) = p + \bar{\alpha}(p, r) \cdot r \in \partial\Omega.$$

Hence, $g(r)$ is on the boundary for $r \neq 0$.

It is noteworthy that this approach allows one to construct a mapping onto a non-convex connected union of convex domains. On the other hand, any method that relies on a convex combination of basis vectors, where the weights are found using the operation *softmax*, allows one to construct points only inside the domain rather than on the boundary. For example, consider the problem of projecting points onto the boundary of a convex set:

$$\begin{gathered}\min \|z - f_\theta(z)\|_p \\ \text{under constraints } f_\theta(z) \in \partial\Omega.\end{gathered} \tag{8}$$

Examples of projection onto a domain specified by a set of linear constraints for the L_1 and L_2 norms are shown in Fig. 4. To solve each of the problems, a neural network was trained, containing 5 layers of size 100, minimizing (8), the set of values of which is specified as $\partial\Omega$. It can be seen from Fig. 6 that the generated points are on the boundary of the constraints.

3. EXPERIMENTS

3.1. Optimization Problems

One of the applications of the proposed approach is solving optimization problems with constraints. For each particular problem, the vector of input parameters z_i is optimized in order to minimize the loss function $l_i(f_\theta(z_i))$. For testing, 50 sets of problems and constraints were generated for output dimensions of 2,

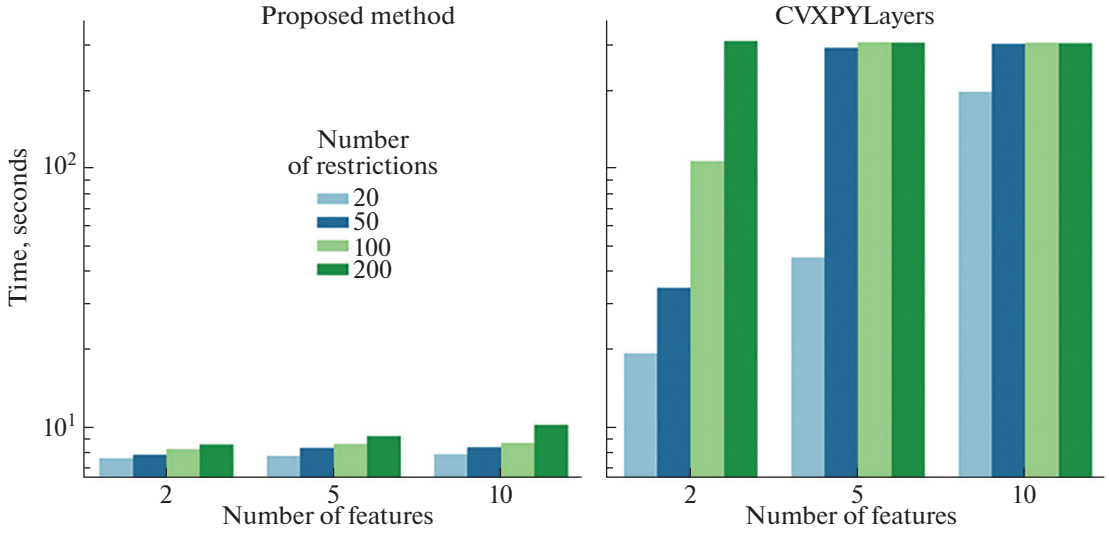

Fig. 5. Comparison of the running time of the proposed neural network and projection using CVXPYLayers.

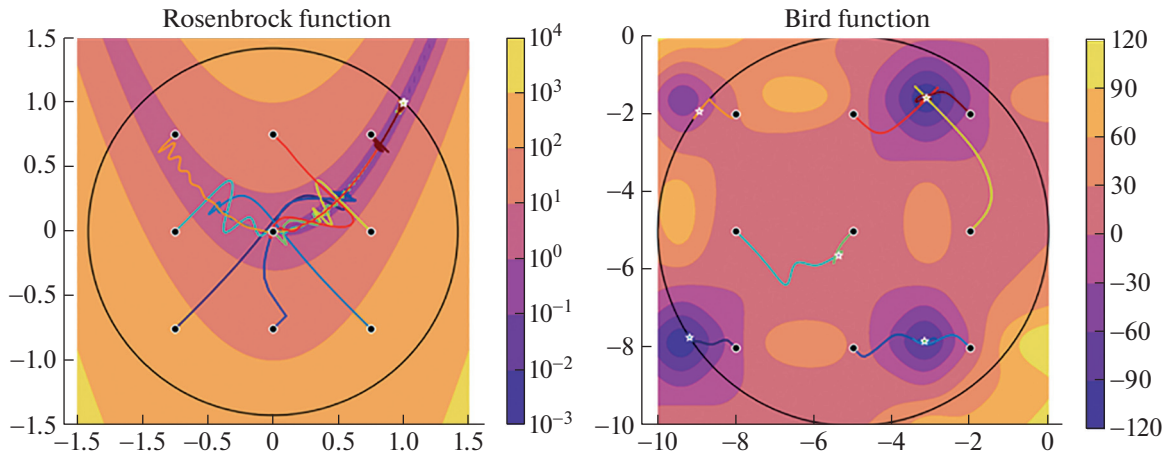

Fig. 6. Optimization trajectories for the Rosenbrock [15] and Bird [16] functions.

5, and 10, with 50, 100, and 200 constraints separately for the cases of linear and quadratic constraints, so that the domain Ω was bounded.

For linear constraints, we generated m vectors $a_1,\ldots,a_m \sim \mathcal{N}(0,I)$ defining points belonging to hyperplanes and normals to these hyperplanes. Then, $b_i = a_i^{\mathrm{T}} a_i$ and $\Omega = \{x \,|\, a_i^{\mathrm{T}} x \le b_i,\ i = 1,\ldots,m\}$. For quadratic constraints, positive semidefinite matrices $P^{(i)}$ and vectors $q_i \sim \mathcal{N}(0,I)$, $i = 1,\ldots,m$, were generated. The constraints were then shifted to satisfy the clearance constraints $b_i > 0$. As a result, we obtain a system of quadratic constraints $\Omega = \{x \,|\, x^{\mathrm{T}} P^{(i)} x + q_i^{\mathrm{T}} x \le b_i,\ i = 1,\ldots,m\}$.

The relative error for comparison of models is calculated from the value of the loss function of the resulting solution, $l_i(f_\theta(x_i))$, relative to the loss function of the reference solution, $l_i(x_i^*)$:

$$RE = \frac{\max\{0, l_i(f_\theta(x_i)) - l_i(x_i^*)\}}{|l_i(x_i^*)|} \cdot 100\%.$$

Reference solutions were obtained using the algorithm *OSQP* [14], designed exclusively for solving linear and quadratic problems.

Table 1 shows the average relative errors for optimization problems with linear (LL) and quadratic (QL) loss functions, as well as linear (LC) and quadratic (QC) constraints. It can be seen from the table that the method allows one to optimize the input parameters r and s for the proposed layer. It can be seen that this layer does not degrade the gradient for the entire neural network.

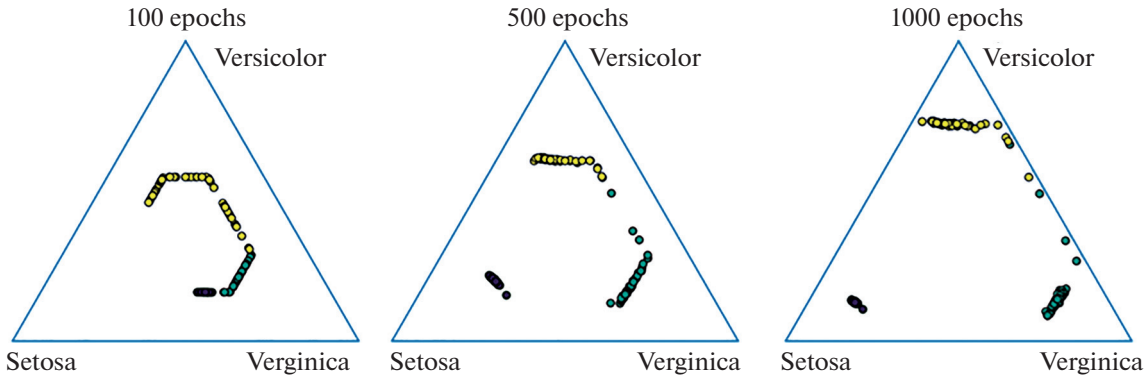

Fig. 7. Class probabilities for different numbers (100, 500, 1000) of training epochs.

For comparison with a neural network, the CVXPYLayers library [7] was chosen, which allows one to specify differentiable optimization layers within the neural network. Figure 5 shows a comparison of the optimization execution time with the same hyperparameters, provided that, after 5 minutes from the start of optimization, the CVXPYLayers algorithm stopped, even if the optimization was not completed. As can be seen from the figure, the proposed algorithm requires significantly less overhead in terms of performance.

The structure of the neural network makes it possible to implement algorithms for solving arbitrary problems of both convex and nonconvex optimization. Figure 6 shows optimization trajectories for the Rosenbrock function [15] with quadratic constraints:

$$\mathscr{L}_{\text{Ros}}(x) = (1 - x_1^2) + 100(x_2 - x_1^2)^2,$$

$$\Omega_{\text{Ros}} = \left\{x \,|\, x_1^2 + x_2^2 \le 2\right\},$$

and the Bird function [16], which has 4 local minima in Ω_{Bird}, two of which lie on the boundary of Ω_{Bird}:

Table 1. *RE* for problems with various loss functions and constraints

m	n	*LL*–*LC*	*LL*–*QC*	*QL*–*LC*	*QL*–*QC*
50	2	3.6×10^{-5}	1.3×10^{-4}	3.9×10^{-5}	7.1×10^{-6}
	5	2.4×10^{-3}	7.7×10^{-5}	1.2×10^{-3}	4.9×10^{-4}
	10	1.4×10^{-2}	4.3×10^{-6}	6.9×10^{-3}	3.5×10^{-3}
100	2	2.6×10^{-5}	1.6×10^{-4}	5.4×10^{-5}	6.0×10^{-6}
	5	2.1×10^{-3}	8.1×10^{-4}	1.6×10^{-3}	8.9×10^{-4}
	10	1.1×10^{-2}	1.3×10^{-5}	8.0×10^{-3}	4.3×10^{-3}
200	2	1.5×10^{-5}	3.1×10^{-4}	5.0×10^{-5}	1.2×10^{-5}
	5	2.6×10^{-3}	1.1×10^{-3}	1.3×10^{-3}	5.9×10^{-4}
	10	1.2×10^{-2}	2.8×10^{-4}	7.7×10^{-3}	4.9×10^{-3}

$$\mathscr{L}_{\text{Bird}}(x) = \sin(x_2)\, e^{(1-\cos(x_1))^2} + \cos(x_1)\, e^{(1-\sin(x_2))^2} + (x_1 - x_2)^2,$$

$$\Omega_{\text{Bird}} = \{x \,|\, (x_1 + 5)^2 + (x_2 + 5)^2 < 25\}.$$

The function $\mathscr{L}_{\text{Ros}}(x)$ has a global minimum at the point (1, 1). The boundaries of Ω_{Ros} and Ω_{Bird} are shown as large black circles. 2000 iterations of the Adam algorithm were used with a learning rate of 0.1; 9 points on a uniform mesh from -0.75 to 0.75 were selected as starting points, for which the optimization trajectories are shown in Fig. 6, where the end point is indicated by a white star.

3.2. Example of Classification

To illustrate the capabilities of the proposed method, we considered a classification problem using as an example the simplest data set "Iris" with constraints on the upper bounds $\overline{p}_i$ for each class probability x_i:

$$f_\theta(z) \in \left\{x \,|\, 0 \le x_i \le \overline{p}_i,\ \mathbf{1}^{\mathrm{T}} x = 1\right\}.$$

These constraints can play a balancing role in learning, reducing the influence of already correctly classified points on the loss function. To illustrate how a neural network learns with these constraints and to make it easier to visualize the results, we chose this classic dataset, containing 3 classes and 150 examples. Three classes allow one to visualize results using a single probability simplex. In this example, we are setting the upper bounds $\overline{p}_i = 0.75$ for $i = 1, 2, 3$. Figure 7 shows unit simplexes and points, which are the output of a neural network trained for 100, 500, and 1000 epochs. The neural network consists of 5 layers of size 64. The colors of the dots indicate the corresponding classes (Setosa, Versicolor, Virginica). It can be seen from Fig. 7 that the constraints make effect not only when the output points are very close to them, but also throughout the entire training of the network.

4. CONCLUSIONS

A method has been presented that imposes hard constraints on the output values of a neural network. The method is extremely simple from a computational point of view, is implemented by an additional layer of a neural network, and allows one to impose a large class of constraints: linear and quadratic inequalities, linear equalities, and joint constraints on inputs and outputs. The method can be directly extended to any convex constraints. It allows one to approximate orthogonal projections onto a convex set or generate vectors on a boundary set.

The disadvantage of the method is that it does not allow working with exclusively conical constraints. In the future, the method proposed can be modified to solve problems with such constraints. In addition, it is interesting to extend the ideas of the proposed method to the case of nonconvex constraints.

An important direction is also the consideration of various machine learning applications that require taking into account constraints, e.g., physics-informed neural networks [17]. Each application can be regarded as a separate problem for further study.

FUNDING

This work was supported by the Russian Science Foundation, grant No. 21-11-00116.

CONFLICT OF INTEREST

The authors of this work declare that they have no conflicts of interest.

REFERENCES

1. P. Marquez-Neila, M. Salzmann, and P. Fua, "Imposing hard constraints on deep networks: Promises and Limitations," in *CVPR Workshop on Negative Results in Computer Vision* (2017), pp. 1–9.
2. T. Frerix, M. Niessner, and D. Cremers, "Homogeneous linear inequality constraints for neural network activations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (2020), pp. 748–749.
3. J. Y. Lee, S. V. Mehta, M. Wick, J.-B. Tristan, and J. Carbonell, "Gradient-based inference for networks with output constraints," *Proceedings of the AAAI Conference on Artificial Intelligence* (*AAAI-19*) (2019), Vol. 33, pp. 4147–4154.
4. P. L. Donti, D. Rolnick, and J. Z. Kolter, "DC3: A learning method for optimization with hard constraints," *International Conference on Learning Representations* (*ICLR*) (2021), pp. 1–17.
5. M. Brosowsky, F. Keck, O. Dunkel, and M. Zollner, "Sample-specific output constraints for neural networks," *The 35th AAAI Conference on Artificial Intelligence* (*AAAI-21*) (2021), pp. 6812–6821.
6. B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," *Proceedings of the 34th International Conference on Machine Learning* (PMLR, 2017), pp. 136–145.
7. A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," Adv. Neural Inf. Process. Syst. **32**, 1–13 (2019).
8. M. Li, S. Kolouri, and J. Mohammadi, "Learning to solve optimization problems with hard linear constraints," IEEE Access **11**, 59995–60004 (2023).
9. R. Balestriero and Y. LeCun, "POLICE: Provably optimal linear constraint enforcement for deep neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing* (*ICASSP*) (IEEE, 2023), pp. 1–5.
10. Y. Chen, D. Huang, D. Zhang, J. Zeng, N. Wang, H. Zhang, and J. Yan, "Theory-guided hard constraint projection (HCP): A knowledge-based data-driven scientific machine learning method," J. Comput. Phys. **445**, 110624 (2021).
11. G. Négiar, M. W. Mahoney, and A. Krishnapriyan, "Learning differentiable solvers for systems with hard constraints," in *The 11th International Conference on Learning Representations* (*ICLR*) (2023), pp. 1–19.
12. J. Kotary, F. Fioretto, and P. Van Hentenryck, "Learning hard optimization problems: A data generation perspective," Adv. Neural Inf. Process. Syst. **34**, 24981–24992 (2021).
13. J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder, "End-to-end constrained optimization learning: A survey," *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (*IJCAI-21*) (2021), pp. 4475–4482.
14. B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," Math. Program. Comput. **12** (4), 637–672 (2020).
15. H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," Comput. J. **3** (3), 175–184 (1960).
16. S. K. Mishra, "Some new test functions for global optimization and performance of repulsive particle swarm method" (2006). https://ssrn.com/abstract=926132.
17. M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," J. Comput. Phys. **378**, 686–707 (2019).