AutoRIC: Automated Neural Network Repairing Based on Constrained Optimization

XINYU SUN* and WANWEI LIU*, National University of Defense Technology, China
SHANGWEN WANG, National University of Defense Technology, China
TINGYU CHEN, National University of Defense Technology, China
YE TAO, National University of Defense Technology, China
XIAOGUANG MAO, National University of Defense Technology, China

Neural networks are important computational models used in the domains of artificial intelligence and software engineering. Parameters of a neural network are obtained via training it against a specific dataset with a standard process, which guarantees each sample within that set is mapped to the correct class. In general, for a trained neural network, there is no warranty of high-level properties, such as fairness, robustness, etc. In this case, one need to tune the parameters in an alternative manner, and it is called repairing. In this paper, we present AutoRIC (*Auto*mated *R*epair w*I*th *C*onstraints), an analytical-approach-based white-box repairing framework against general properties that could be quantitatively measured. Our approach is mainly based on constrained optimization, namely, we treat the properties of neural network as the optimized objective described by a quadratic formula about the faulty parameters. To ensure the classification accuracy of the repaired neural network, we impose linear inequality constraints to the inputs that obtain incorrect outputs from the neural network. In general, this may generate a huge amount of constraints, resulting in the prohibitively high cost in the problem solving, or even making the problem unable to be solved by the constraint solver. To circumvent this, we present a selection strategy to diminish the restrictions, i.e., we always select the most 'strict' ones into the constraint set each time. Experimental results show that repairing with constraints performs efficiently and effectively. AutoRIC tends to achieve a satisfactory repairing result whereas brings in a negligible accuracy drop. AutoRIC enjoys a notable time advantage and this advantage becomes increasingly evident as the network complexity rises. Moreover, experiment results also demonstrate that repairing based on unconstrained optimizations are not stable, which embodies the necessity of constraints.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Software and its engineering** → **Software defect analysis**; • **Theory of computation** → **Quadratic programming**.

Additional Key Words and Phrases: Deep Neural Networks, Repair, Constrained Optimization

ACM Reference Format:
Xinyu Sun, Wanwei Liu, Shangwen Wang, Tingyu Chen, Ye TAO, and Xiaoguang Mao. 2024. AutoRIC: Automated Neural Network Repairing Based on Constrained Optimization. *ACM Trans. Softw. Eng. Methodol.* 37, 4, Article 111 (August 2024), 29 pages. https://doi.org/10.1145/xxxxxxx.xxxxxxx

*Both authors contributed equally to this research.

Authors' addresses: Xinyu Sun, sunxinyu17@nudt.edu.cn; Wanwei Liu, wwliu@nudt.edu.cn, National University of Defense Technology, Hunan, Changsha, China, 410003; Shangwen Wang, National University of Defense Technology, Hunan, China, wangshangwen13@nudt.edu.cn; Tingyu Chen, National University of Defense Technology, Hunan, China, chentingyu@nudt.edu.cn; Ye TAO, National University of Defense Technology, Hunan, China, taoye0117@nudt.edu.cn; Xiaoguang Mao, National University of Defense Technology, Hunan, China, xgmao@nudt.edu.cn.

1049-331X/2024/8-ART111 $15.00
https://doi.org/10.1145/xxxxxxx.xxxxxxx

1 INTRODUCTION

Neural networks have found success in various domains such as image recognition [23], natural language processing [20], medical diagnosis [21], aircraft collision avoidance [19], and autonomous driving [6]. Despite their achievements, neural networks are not flaw-free. Instances of errors in neural networks have, in some cases, resulted in serious consequences, including loss of life [16] and wrongful arrests [17, 18]. Hence, much like the process of debugging in traditional software programs [3], there is a critical need to devise automated methods for rectifying these defects in neural networks and ensuring that they adhere to desired standards.

Efforts to rectify unexpected behaviors in neural networks typically lean towards retraining with supplementary data [28, 29]. While retraining is a natural and often effective approach, it can face challenges in real-world applications due to its potential difficulty and cost [15]. For example, training a convolutional image recognition neural network has been reported to take days or even weeks [39]. Moreover, the original training dataset may not be accessible for retraining, as it could involve private medical information, sensitive intellectual property, or even be lost. Additionally, retraining has the potential to introduce arbitrary changes to the neural networks, possibly leading to the emergence of new bugs in its behavior.

To tackle these issues, researchers have recently put forward a number of approaches which aim at adjusting the parameters of an existing neural network. This allows for the elimination of unexpected behaviors, the preservation of other functionalities, all while bypassing the need for retraining. Specifically, CARE [41] first utilizes causality analysis to locate the parameters whose values need to be modified and then searches for suitable parameter values using a Particle Swarm Optimisation (PSO) algorithm. Such a search-based process is treated as a "black-box". Another area of research emphasizes the concept of *white-box repair*, which reformulate the problem with standard mathematical models, such as linear programming or constraint solving [39, 42], namely the whole repairing process are explicitly formulated in mathematical manner. Notably, these work typically focus on some specific properties of the neural network. For instance, both PRDNN [39] and APRNN [42] are specialized for the qualitative properties such as safety, while other critical quantitative properties such as the fairness are understudied. Fairness is a desirable property in neural networks used for applications with societal significance. For example, Google's photo tagger produced offensive labels, classifying black people into"gorillas" [44]. FAD [1], Ethical Adversaries [9] and FairNeuron [13] that try to provide such functionality. Based on the observation that optimizing for accuracy and fairness can be conflicting goals during training, these frameworks introduce an adversary to monitor and enhance the fairness of the training process.

In this paper, we present an analytical-approach-based white-box neural network repair framework that against general properties. Compared to heuristic algorithms, the optimization objectives and the relationships with parameters in AutoRIC are presented in the form of expressions. The core part of this approach is also to convert the neural network repair task into a constrained optimization problem, yet it can deal with any properties that could be quantitatively measured. To ensure the accuracy of the repaired network, we also impose linear constraints which describe the classification specifications of the original network to the optimization. However, this would generate a huge amount of constraints in general, and that would severely hinder the scalability of the repairing — or even worse, sometimes we could not get a solution from the solver. To circumvent this, our approach incorporates a constraint-selection module aimed at reducing the total number of constraints. To this end, we select those pose the greatest difficulty in terms of correction, and use them as the primary focus for constraint solving. The underlying intuition is that by addressing these strong constraints, other inputs that are comparatively easier to correct could also achieve accurate results through the process of constraint solving. Henceforth, we successfully model the

repair problem with a quadratic programming problem (the subsequent experiments reveal that with the same fairness increase, quadratic programming has indeed less accuracy decrease than linear programming,), and to the best of our knowledge we design different ways for addressing both convex and non-convex objects, and thus can obtain a repaired network.

The proposed approach has been experimentally evaluated via conducting a series of experiments. For fairness tasks, AutoRIC has an average 63.79% and 2.89% fairness improvement and accuracy decrease. It is worth mentioning that our approach can decrease the fairness of Bank to 0 with only 1% accuracy drop. For robustness tasks, AutoRIC improves the robustness radius of 2-Conv and 3-Linear neural network from 0.126 to 0.140, with 15.87% improvement and only 1.01% accuracy decrease. Further, results show that AutoRIC is between 2 and 7 times more efficient than CARE and PVNN in fairness tasks.

Main contributions of this paper can be summarized as follows:

- First of all, to the best of our knowledge, AutoRIC is the first white-box repair method based on an analytical approach. Based on the sampling-fitting process, as well as the classification constraint extracting, we provide a uniform way that can convert the repair task (such as fairness, robustness) into a quadratic programming problem, provided that the goal of repair can be quantitatively measured.
- Second, since the (linear) constraints of that quadratic programming are generated from data classifications, they usually constitute a huge set. We in this paper reveal a general principle of constraint-selection, which may significantly reduce the amount of linear constraints.
- Based on the proposed approach, we have implemented a prototype, called AutoRIC. Extensive experiments have demonstrated the superior effectiveness and efficiency of the approach. By prioritizing network repair, AutoRIC enjoys a notable time advantage. This advantage becomes increasingly evident as the network complexity rises.

The reminder of this paper is organized as follows: In Sect. 2, necessary notions and notations are introduced. In Sect. 3, we elaborate how the repairing problem is formulated by a (constrained) quadratic programming, and describe how it is solved. Sect. 5 provides the experimental results of our repair approach. We list some related work in Sect. 7, and we conclude this paper with Sect. 9.

2 BACKGROUND

2.1 Kullback-Leibler divergence

Kullback-Leibler divergence (KL divergence for short)[24], also known as relative entropy, is a measure of the difference between two probability distributions $\boldsymbol{P}$ and $\boldsymbol{Q}$. It is a non-symmetric measure and is defined as follows:

For discrete probability distributions

$$D_{KL}(\boldsymbol{P} \parallel \boldsymbol{Q}) = \sum_i \boldsymbol{P}(i) \log \frac{\boldsymbol{P}(i)}{\boldsymbol{Q}(i)},$$

and for continuous probability distributions

$$D_{KL}(\boldsymbol{P} \parallel \boldsymbol{Q}) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x.$$

Here, $\boldsymbol{P}$ represents the true distribution, and $\boldsymbol{Q}$ represents the theoretical or approximate distribution. The KL divergence from $\boldsymbol{Q}$ to $\boldsymbol{P}$ is always non-negative and is zero if and only if $\boldsymbol{P}$ and $\boldsymbol{Q}$ are identical. It quantifies the amount of "information lost" when one uses the distribution $\boldsymbol{Q}$ to approximate the distribution $\boldsymbol{P}$. The KL divergence, in this context, can be thought of as a measure of the difference in disorder between two systems, with $\boldsymbol{P}$ and $\boldsymbol{Q}$ representing different states of a system.

2.2 Optimizations

A set $C \subseteq \mathbb{R}^n$ is *convex* if

$$\boldsymbol{x} \in C \wedge \boldsymbol{y} \in C \wedge 0 \leq \theta \leq 1 \implies \theta\boldsymbol{x} + (1-\theta)\boldsymbol{y} \in C$$

holds. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *convex*, if $\operatorname{dom} f$ is a convex set and

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y}), \qquad 0 \leq \theta \leq 1$$

for each $\boldsymbol{x}, \boldsymbol{y} \in \operatorname{dom} f$.

If f is first-order differentiable, we let $\nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)^{\mathrm{T}}$, call it the *gradient* of f at $\boldsymbol{x}$, where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}} \in \mathbb{R}^n$. If f is further second-order derivable, then we denote the matrix $\left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right)_{n \times n}$ by $\nabla^2 f(\boldsymbol{x})$, and call it the *Hessian matrix* of f at $\boldsymbol{x}$.

Theorem 1. *Suppose that f is second order derivable, then the followings are pairwise equivalent:*

(1) f is convex;
(2) $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + (\boldsymbol{y}-\boldsymbol{x})^{\mathrm{T}} \nabla f(\boldsymbol{x})$ for every $\boldsymbol{x}, \boldsymbol{y} \in \operatorname{dom} f$;
(3) $\nabla^2 f(\boldsymbol{x})$ is semi-positive definite (SPD) for every $\boldsymbol{x} \in \operatorname{dom} f$.

We say that a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is *semi-positive definite* (SPD, for short) only if $\boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x} \geq 0$ for every $\boldsymbol{x} \in \mathbb{R}^n$.

An *optimization problem* (a.k.a., *programming*) $\mathcal{P}$ can be written in the canonical form

$$\min f(\boldsymbol{x}) \qquad \text{s.t.} \qquad \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^m$, they are called the *goal* and the *constraints* of the optimization problem, respectively. Let us denote $\operatorname{dom} f \cap \operatorname{dom} \boldsymbol{g}$ by $\operatorname{dom} \mathcal{P}$ in what follows. For some $\boldsymbol{x}_0 \in \operatorname{dom} \mathcal{P}$, if

$$f(\boldsymbol{x}_0) \leq f(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \operatorname{dom} \mathcal{P}$$

holds, then we say that $\boldsymbol{x}_0$ is the *(globally) optimal value* of $\mathcal{P}$. Otherwise, if there exists some Euclidean ball $B_N(\boldsymbol{x}_0, r) \subseteq \operatorname{dom} \mathcal{P}$ making

$$f(\boldsymbol{x}_0) \leq f(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in B_N(\boldsymbol{x}_0, r)$$

then we say that $\boldsymbol{x}_0$ are a *locally optimal value* of $\mathcal{P}$ — recall that

$$B_N(\boldsymbol{x}_0, r) = \begin{cases} \{(x_1, x_2, \ldots, x_n)^{\mathrm{T}} \mid \sum_i \operatorname{abs}(x_i - x_{0,i}) \leq r\}, & N = 1 \\ \{\boldsymbol{x} \mid \sqrt{(\boldsymbol{x}-\boldsymbol{x}_0)^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{x}_0)} \leq r\}, & N = 2 \\ \{(x_1, x_2, \ldots, x_n)^{\mathrm{T}} \mid \max\{\operatorname{abs}(x_i - x_{0,i})\}_i \leq r\}, & N = \infty \end{cases},$$

where $\boldsymbol{x}_0 = (x_{0,1}, x_{0,2}, \ldots, x_{0,n})^{\mathrm{T}}$.

Convex Optimization Problems. For an optimization problem $\mathcal{P}$ with convex domain $\operatorname{dom} \mathcal{P}$, goal f and constraints $\boldsymbol{g}$, if f is convex and in addition $\boldsymbol{g}$ is also convex, namely

$$\boldsymbol{g}(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leq \theta \boldsymbol{g}(\boldsymbol{x}) + (1-\theta)\boldsymbol{g}(\boldsymbol{y}), \qquad 0 \leq \theta \leq 1$$

holds for every $\boldsymbol{x}, \boldsymbol{y} \in \operatorname{dom} \mathcal{P}$, then $\mathcal{P}$ is said to be *convex*.

Theorem 2. *If $\boldsymbol{x}_0$ is a locally optimal value of the convex optimization problem $\mathcal{P}$, then it must be the globally optimal value of $\mathcal{P}$.*

Quadratic Optimization Problems. The convex optimization problem is called a *quadratic programming* (QP for short) if the objective function is (convex) quadratic, and the constraint functions are affine. Below is an example of this problem.

Example 1. *Let $f(\boldsymbol{x})$ be the quadratic polynomial $\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{x}+\boldsymbol{p}^{\mathrm{T}}\boldsymbol{x}+c$ where $\boldsymbol{Q}\in\mathbb{R}^{n\times n}$ is a symmetric matrix, $\boldsymbol{p}\in\mathbb{R}^n$ and $c\in\mathbb{R}$. Then f is convex if and only if $\boldsymbol{Q}$ is SPD, because $\nabla^2 f(\boldsymbol{x})=\boldsymbol{Q}$.*

It is well known that LP problem belongs to the P class and can be solved in polynomial time. However, for the QP problem, when the coefficient matrix $\boldsymbol{Q}$ is positive definite, the QP problem belongs to the P class and can be solved in polynomial time. When $\boldsymbol{Q}$ is not positive definite, the QP problem is NP-Hard.

Numerous mature approaches have been developed for both convex and non-convex optimizations. Below lists some typical examples.

- In the case that the goal is a convex quadratic formula and all constraints are linear, the problem is called to be *semi-definite programming* (SPD for short). One can use the *active set method* [4], *Lagrangian Dual Method*[34], or use the *interior point approach* [10] to obtain the optimum.
- Particularly, when that quadratic formula becomes a linear one (i.e., the second-order coefficient matrix is $\mathbf{0}$), it is called *linear programming* (LP for short). Besides the aforementioned methods, one can also use the *simplex method* [8], which is considered to be more practical.
- Whenever the coefficient matrix (i.o.w., the Hessian matrix) is not SPD in the quadratic programming, one can perform the so-called *sequential convex approximation* approach to accomplish the optimization. Alternatively, one can also use the approximate manner to deal with it: i.e., let $\boldsymbol{Q}$ be the corresponding matrix, then one just need to replace it with a SPD matrix $\boldsymbol{Q}+\lambda\boldsymbol{I}$ for some $\lambda>0$.

In this paper, we are particularly concerned about quadratic programming, since it can describe the goal more accurately in comparison to LP, and as a result, our optimization problem can also be solved effectively with many powerful optimization techniques. The experiments indicate that completing the same repairing tasks, QP exhibits less accuracy decrease than LP.

2.3 Neural Networks and Repairing

Neural Networks. Classification neural network can be viewed as a mapping from the feature space to a finite set of classes (a. k. a., classification space). Typically, a classification neural network can be decomposed into several layers, and each layer corresponds to a composition of an affine mapping and an activation function (which is in general non-affine). Hence, an (n+1)-layer neural network $\mathcal{N}$ can be uniquely determined via a series of parameters

$$(\boldsymbol{M}_1,\boldsymbol{b}_1),(\boldsymbol{M}_2,\boldsymbol{b}_2),\ldots,(\boldsymbol{M}_n,\boldsymbol{b}_n)$$

where $\boldsymbol{M}_i$ is the *weight matrix* between the i-th and the $i+1$-th layer, and $\boldsymbol{b}_i$ is the i-th layer's *bias vector*. Then, for an input vector $\boldsymbol{s}$, we may obtain a series of vectors $\boldsymbol{s}_1,\boldsymbol{s}_2,\ldots,\boldsymbol{s}_{n+1}$ where $\boldsymbol{s}_1=\boldsymbol{s}$ and $\boldsymbol{s}_{i+1}=\sigma_i(\boldsymbol{M}_i\boldsymbol{s}_i+\boldsymbol{b}_i)$. In the above, σ_i is the activation function if $i<n$, and σ_n is the softmax mapping. Suppose that $\boldsymbol{s}_{n+1}=(w_1,\ldots,w_m)^{\mathrm{T}}$, we let

$$\mathcal{N}(\boldsymbol{s})=\arg\max_i w_i$$

as the output. Typically, an activation function might be

- the sigmoid function, defined as $\mathrm{sigmoid}(x)=\dfrac{1}{1+\mathrm{e}^{-x}}$;
- the relu function, defined as $\mathrm{relu}(x)=\max\{x,0\}$;

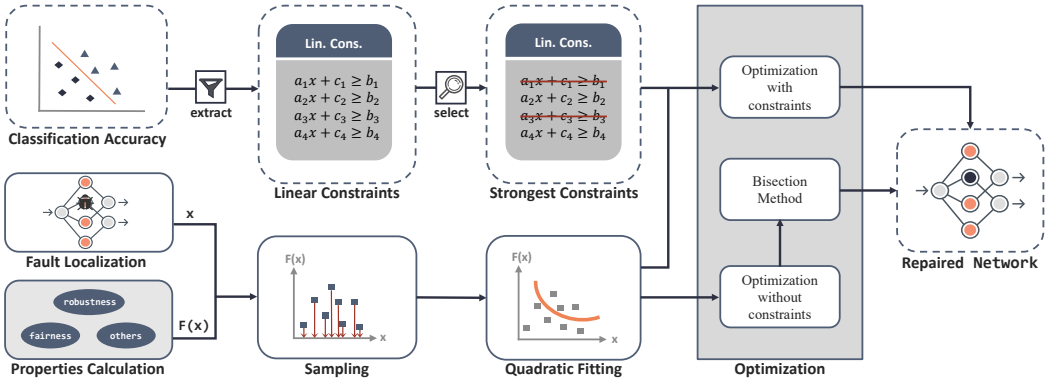


Fig. 1. The workflows of AutoRIC

- the tanh function, defined as $\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

and so on. For the activation function σ, we let $\sigma(\boldsymbol{c}) = (\sigma(c_1), \ldots, \sigma(c_n))^{\mathrm{T}}$ if $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)^{\mathrm{T}}$. In addition, we let $\mathrm{softmax}(\boldsymbol{c}) = \left(\dfrac{e^{c_1}}{\sum_{i=1}^n e^{c_i}}, \dfrac{e^{c_2}}{\sum_{i=1}^n e^{c_i}}, \ldots, \dfrac{e^{c_n}}{\sum_{i=1}^n e^{c_i}} \right)^{\mathrm{T}}$, which normalizes a vector to a discrete distribution.

Neural Network Repairing. Given a neural network $\mathcal{N}$, together with a series of designated parameters $\boldsymbol{y} = \{y_1, y_2, \ldots, y_n\}$ of $\mathcal{N}$ (namely, each y_i can be some weight with some bias), and a set of labeled (a.k.a., classified) samples $\{(\boldsymbol{s}_i, c_i)\}$ having $\mathcal{N}(\boldsymbol{s}_i) = c_i$, the goal of repair is to optimize some properties φ related to $\mathcal{N}$ by tuning the parameters without changing the label of each of samples (or, for a relaxed-version, make the change as small as possible).

In general, a network corresponds to a non-linear mapping from the input space to the collection of (discrete) classes. As a result, the constraints that maintain each $\mathcal{N}(\boldsymbol{s}_i)$ unchanged is in general non-linear, or even non-convex. Retraining alters the parameters globally and is typically used to improve the accuracy of a model, rather than enhancing specific properties of the model. The properties and accuracy of a model are usually orthogonal to each other. Repairing involves making local adjustments to the parameters, while ensuring that the model's accuracy does not decrease significantly under the condition of repairing the properties.

3 AUTORIC: A FITTING-BASED REPAIR FRAMEWORK

We now consider the problem of neural network repairing with accuracy constraints, namely, to enhance some (quantized) properties without changing the classification on the given dataset. The overall workflow are demonstrated in Figure 1. First, we introduce the preparatory work before repairing networks. On the one hand, beginning with the fault localization of trained neural networks, we choose the faulty parameters $\boldsymbol{y}$. Following it, we prepare to calculate the concerned properties of neural networks with $f(\boldsymbol{y})$, e.g. robustness and fairness. Then, we sample the input data and calculate corresponding properties for certain trained network. Meanwhile, we record the value of the faulty parameters. The value of faulty parameters and the quantification of property can combine as a sampled point. Subsequently, we employ these points to do quadratic fitting to establish the goal to be optimized. On the other hand, to guarantee the classification accuracy, we extract linear constraints from trained neural networks. Then, we may regard the repairing problem as an optimization problem.

First of all, given a neural network $\mathcal{N}$ and a specification φ, we may take one of the following fault localization approaches to determine the decision variables $\boldsymbol{y}$.

- The causality-analysis based approach, presented in [41], which may effectively locate buggy parameters with a designated scale.
- DeepFault [11] — the first deep neural network white-box testing method based on fault localization.
- A more fine-gained approach which locates the faulty weights of neurons presented in [47].

Since we compare our work mainly with CARE, we pay more attention to the first approach. Basically, such analysis aims at finding the parameters affecting the property most significantly, and it uses the structural causal models (SCMs) to perform such detection. One can consider this part as a black-box, and we mainly address how to describe the optimization problem and how to solve it.

3.1 Establish the Goal of Optimization

Once the buggy parameters are selected, we can explicitly partition the parameters into the variables and constants, and respectively encode these two sets with the vectors $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^{\mathrm{T}}$ and $\boldsymbol{c} = (c_1, c_2, \ldots, c_m)^{\mathrm{T}}$ where the amount of variables is n and the amount of constants is m. Now we intend to formulate the goal that needs to be repaired in terms of $\boldsymbol{y}$, namely, we need to find an analytical expression $f(\boldsymbol{y})$ to describe the goal. Let $\boldsymbol{y}^{(0)} = (y_1^{(0)}, y_2^{(0)}, \ldots, y_n^{(0)})^{\mathrm{T}}$ be the original value of $\boldsymbol{y}$ before repairing, using Taylor's formula, within a small neighborhood, we have

$$f(\boldsymbol{y}) = c_0 + \boldsymbol{p}_0^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{y}_0) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{y}_0)^{\mathrm{T}}\boldsymbol{Q}_0(\boldsymbol{y} - \boldsymbol{y}_0) + o(\|\boldsymbol{y} - \boldsymbol{y}^{(0)}\|)$$

where $c_0 = f(\boldsymbol{y}^{(0)})$, $\boldsymbol{p}_0 = \nabla f(\boldsymbol{y}^{(0)})$ and $\boldsymbol{Q}_0 = \nabla^2 f(\boldsymbol{y}^{(0)})$ — this enlightens us that f can be approximately represented via some quadratic polynomial.

To determine the coefficients like c, $\boldsymbol{p}$ and $\boldsymbol{Q}$, we need sample a set of points

$$P = \left\{\left(\boldsymbol{y}^{(i)}, z^{(i)} = f(\boldsymbol{y}^{(i)})\right)\right\}_{i=0}^{m} \subseteq \mathbb{R}^{n+1}$$

with some proper amount m. Let

$$f(\boldsymbol{y}) \approx \hat{f}(\boldsymbol{y}) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{y}_0)^{\mathrm{T}}\boldsymbol{Q}_0(\boldsymbol{y} - \boldsymbol{y}_0) + \boldsymbol{p}_0^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{y}_0) + c_0 = \frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{y} + \boldsymbol{p}^{\mathrm{T}}\boldsymbol{y} + c$$

be the approximation of $f(\boldsymbol{y})$, then the coefficients $\boldsymbol{Q}$, $\boldsymbol{p}$ and c should minimize the error

$$d = \sum_{i=1}^{m} \left\| \hat{f}(\boldsymbol{y}^{(i)}) - z^{(i)} \right\|_2^2.$$

It is worth noting that when the quadratic matrix $\boldsymbol{Q}$ is indefinite, we consider the quadratic matrix $\lambda\boldsymbol{I} + \boldsymbol{Q}$ to approximate $\boldsymbol{Q}$. By appropriately choosing the value λ, we can make it positive definite. To minimize the correction bias, we set

$$\lambda = \min(\mathrm{spec}(\boldsymbol{Q})).$$

That is, the minimum eigenvalue of $\boldsymbol{Q}$.

Suppose that $\boldsymbol{y}^{(i)} = (y_1^{(i)}, y_2^{(i)}, \ldots, y_n^{(i)})$ for each i, and let $\boldsymbol{Q} = \left(q_{j,k}\right)_{n\times n} \in \mathbb{R}^{n\times n}$ with $q_{j,k} = q_{k,j}$, $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)^{\mathrm{T}} \in \mathbb{R}^n$ and $c \in \mathbb{R}$ be the decision variables. Then, from the standard theory of

optimization, let $\frac{\partial d}{\partial q_{j,k}} = 0$, $\frac{\partial d}{\partial p_j} = 0$ and $\frac{\partial d}{\partial c} = 0$, we have

$$\begin{cases} \sum_{i=1}^{m} \Delta^{(i)} \cdot y_j^{(i)} y_k^{(i)} &= 0 \\ \sum_{i=1}^{m} \Delta^{(i)} \cdot y_j^{(i)} &= 0 \\ \sum_{i=1}^{m} \Delta^{(i)} &= 0 \end{cases} \tag{1}$$

for each $0 \le i \le n$, $1 \le j, k \le n$, where $\Delta^{(i)} = (\boldsymbol{y}^{(i)})^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{y}^{(i)} + 2\boldsymbol{p}^{\mathrm{T}} \boldsymbol{y}^{(i)} + 2c - 2z^{(i)}$.

Let

$$\boldsymbol{q} = (q_{1,1}, \ldots, q_{1,n}, q_{2,1}, \ldots, q_{2,n}, \ldots, q_{n,1}, \ldots, q_{n,n})^{\mathrm{T}},$$

and let $\boldsymbol{v} = (\boldsymbol{q}^{\mathrm{T}} \vdots \boldsymbol{p}^{\mathrm{T}} \vdots c)^{\mathrm{T}}$ be the juxtaposition of all variables, by taking the fact that $\Delta^{(i)}$ is essentially

$$\sum_{1 \le j,k \le n} y_j^{(i)} y_k^{(i)} q_{j,k} + 2 \sum_{j=1}^{n} y_j^{(i)} p_j + 2c - 2z^{(i)},$$

thus (1) is a linear system w.r.t. $\boldsymbol{v}$. From which, we may obtain the optimal fitting of $\boldsymbol{Q}$, $\boldsymbol{p}$ and c, respectively.

From the above, we can see that $\hat{f}$ relies on the sampling set P. Further, this set is strongly related to the property to be repaired. Below provides some examples.

Example 2. *Suppose, we are going to repair the group fairness [41] of some input feature — w.l.o.g., just assume it corresponds to the first input x_1, and for simplicity, we further assume that $x_1 \in \{0, 1\}$, and let the classification space $\mathcal{Y} = \{1, 2, \ldots, k\}$. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ range over the input space $\mathcal{X}$, and let*

$$\mathcal{X}_t = \{(x_1, x_2, \ldots, x_n) \in \mathcal{X} \mid x_1 = t\}$$

for $t = 0, 1$, then choose some proper number m, the sampling set $P = \{(\boldsymbol{y}^{(i)}, z^{(i)}) \mid i \le m\}$ is constructed in the following process for each $i \le m$:

(1) If $i \ne 0$, we make a random perturbation of $\boldsymbol{y}^{(0)}$, and let it be $\boldsymbol{y}^{(i)}$.

(2) Then, uniformly sample two sets $S_0^{(i)}$ and $S_1^{(i)}$ within $\mathcal{X}_0$ and $\mathcal{X}_1$, respectively.

(3) Let $\mathcal{N}_{\boldsymbol{y}^{(i)}}$ be the network via changing $\boldsymbol{y}^{(0)}$ to $\boldsymbol{y}^{(i)}$, and then we obtain two categorical distributions $\theta_0^{(i)}$ and $\theta_1^{(i)}$, defined as

$$\theta_j^{(i)}(\ell) = \frac{\#\{\boldsymbol{x} \in S_j^{(i)} \mid \mathcal{N}_{\boldsymbol{y}^{(i)}}(\boldsymbol{x}) = \ell\}}{\# S_j^{(i)}}$$

where $\ell \in \{1, 2, \ldots, k\}$ and $j = 0, 1$.

(4) Finally, let $z^{(i)}$ be the Kullback-Leibler divergence of $\theta_0^{(i)}$ and $\theta_1^{(i)}$, namely

$$z^{(i)} = \mathrm{KL}(\theta_0^{(i)} \| \theta_1^{(i)}) = \sum_{\ell=1}^{k} \theta_0^{(i)}(\ell) \ln \left(\frac{\theta_0^{(i)}(\ell)}{\theta_1^{(i)}(\ell)} \right),$$

and thus a tuple $(\boldsymbol{y}^{(i)}, z^{(i)})$ within P is obtained.

Indeed, it can be extended to the general case that the range of x_1 is much larger, see Sect. 5 for a more illustrative example.

We use $z^{(i)}$ to quantify the fairness of the trained model with respect to a specific feature. On the one hand, a smaller $z^{(i)}$ indicates a smaller discrepancy between the two distributions, signifying greater fairness of the model. When $z^{(i)} = 0$, the distributions are identical, implying that the model is absolutely fair with respect to that feature. On the other hand, the typical form of a convex optimization

problem is to minimize a convex function subject to a set of convex constraints. Therefore, the smaller the value representing fairness, the higher the fairness of the network.

Example 3. *We now consider a special use of repair for improving the robustness at some specific input $\boldsymbol{x}_0 \in \mathcal{X}$. Then, let m be the number of samples, the set P is constituted with the points $(\boldsymbol{y}^{(i)}, z^{(i)})$, and each point is obtained via the following process:*

(1) $\boldsymbol{y}^{(i)}$ is also generated from $\boldsymbol{y}^{(0)}$ by imposing some random perturbation.
(2) Let $r^{(i)}$ be the robustness radius w.r.t. $\mathcal{N}_{\boldsymbol{y}^{(i)}}$, which could be detected in the bisectional manner.
(3) Finally, let $z^{(i)} = \frac{1}{r^{(i)}}$. Here we use the reciprocal of $r^{(i)}$ to be $z^{(i)}$, because we intend to minimize the goal when performing optimization.

Note that for the above two examples, we actually have an implicit constraint, namely, $\hat{f}(\boldsymbol{y}) > 0$, yet it is not in general linear. We will show how to express it in terms of linear expressions in the next subsection.

3.2 Generate Linear Constraints

Suppose that the network $\mathcal{N}$ is constituted by the parameters $\{(\boldsymbol{M}_i, \boldsymbol{b}_i)\}_{i=0}^{n}$ and these parameters (weights in the matrices and biases in the vectors) are re-partitioned into two sets:

- One set consists of the parameters that do not need to be repaired, call such parameters *constants*.
- The other set contains parameters that actually required to be repaired, and each of such parameter is called a *variable*.

So simplify notation, in this section, we use two vectors $\boldsymbol{c}$ and $\boldsymbol{y}$ to represent these two sets, respectively. Namely, we will arrange in order among these parameters.

Since $\boldsymbol{y}$ is viewed as a vector of variables, it may be assigned to concrete values. Then, we use $\mathcal{N}_{\boldsymbol{b}}$ to denote the network modified from $\mathcal{N}$ via assigning $\boldsymbol{y}$ to $\boldsymbol{b}$ for each $\boldsymbol{b} \in \mathbb{R}^n$. Meanwhile, we interchangeably write $\mathcal{N}$ as $\mathcal{N}_{\boldsymbol{y}}$ sometimes to address the variables — just beware that $\mathcal{N}$ is a mapping depending on $\boldsymbol{y}$.

Let $\mathcal{S}$ be the (finite) set of labeled samples, and $\boldsymbol{b}_0$ be the initial value of $\boldsymbol{y}$, then ideally we require that $\mathcal{N}(\boldsymbol{x}) = \mathcal{N}_{\boldsymbol{b}_0}(\boldsymbol{x})$ for every $\boldsymbol{x} \in \mathcal{S}$ when doing such repair. Practically, it just need to guarantee

$$\#\{\boldsymbol{x} \in \mathcal{S} \mid \mathcal{N}(\boldsymbol{x}) \neq \mathcal{N}_{\boldsymbol{b}_0}(\boldsymbol{x})\} < C \tag{2}$$

for some specified tolerant number C. The value of C depends on the particular dataset and the corresponding network. Now, we need to express this in terms of linear constraints about $\boldsymbol{y}$.

Let $\widehat{\mathcal{N}}$ be mapping which is almost identical to $\mathcal{N}$ but without applying softmax and/or arg max to the last step. In addition, for each $i \leq k$, let $\mathcal{N}^{(i)}$ be the mapping defined as

$$\mathcal{N}^{(i)}(\boldsymbol{x}) = \boldsymbol{e}_i^{\mathrm{T}} \widehat{\mathcal{N}}(\boldsymbol{x})$$

where k is the cardinal of the classification space, and $\boldsymbol{e}_i$ is the one-hot vector only its i-th element is 1. Definitely, for each $\boldsymbol{x} \in \mathcal{X}$, we have

$$\mathcal{N}(\boldsymbol{x}) = i \quad \text{iff} \qquad \mathcal{N}^{(i)}(\boldsymbol{x}) \geq \mathcal{N}^{(j)}(\boldsymbol{x}) \text{ for each } j \neq i.$$

Thus, we convert $\mathcal{N}$ with a series of continuous (even, differentiable) functions. With Taylor's expansion, we have

$$\mathcal{N}^{(i)}(\boldsymbol{x}) \approx \mathcal{N}^{(i)}_{\boldsymbol{b}_0}(\boldsymbol{x}) + (\boldsymbol{y} - \boldsymbol{b}_0)^{\mathrm{T}} \left(\nabla_{\boldsymbol{y}} \mathcal{N}^{(i)}(\boldsymbol{x})\right)\Big|_{\boldsymbol{y}=\boldsymbol{b}_0} \tag{3}$$

when viewing $\boldsymbol{y}$ as variables (3) is really a linear expression about $\boldsymbol{y}$, because $\left(\nabla_{\boldsymbol{y}} \mathcal{N}^{(i)}(\boldsymbol{x})\right)\Big|_{\boldsymbol{y}=\boldsymbol{b}_0}$ is actually a constant vector when $\boldsymbol{x}$ and $\boldsymbol{b}_0$ are given.

Let us define $\mathcal{D}_{\boldsymbol{x}}^{(i,j)}(\boldsymbol{y})$ as

$$\mathcal{N}^{(i)}(\boldsymbol{x}) - \mathcal{N}^{(j)}(\boldsymbol{x}),$$

expand into

$$\mathcal{N}_{\boldsymbol{b}_0}^{(i)}(\boldsymbol{x}) + (\boldsymbol{y}-\boldsymbol{b}_0)^{\mathrm{T}}\left(\nabla_{\boldsymbol{y}}\mathcal{N}^{(i)}(\boldsymbol{x})\right)\Big|_{\boldsymbol{y}=\boldsymbol{b}_0} - \mathcal{N}_{\boldsymbol{b}_0}^{(j)}(\boldsymbol{x}) - (\boldsymbol{y}-\boldsymbol{b}_0)^{\mathrm{T}}\left(\nabla_{\boldsymbol{y}}\mathcal{N}^{(j)}(\boldsymbol{x})\right)\Big|_{\boldsymbol{y}=\boldsymbol{b}_0},$$

if $\mathcal{N}(\boldsymbol{x}) = i$ then $\mathcal{D}_{\boldsymbol{x}}^{(i,j)}(\boldsymbol{y})$ is expected to be non-negative for every $j \neq i$, and this is a linear constraint about $\boldsymbol{y}$. We in what follows call linear constraints being of the form $\mathcal{D}_{\boldsymbol{x}}^{(i,j)}(\boldsymbol{y}) \geq 0$ *classification constraints*.

In some special cases, we might have an implicit constraint like $\hat{f}(\boldsymbol{y}) > 0$ (cf. Example 2 and 3). Remember that such a function is fitted from a set $P = \{(\boldsymbol{y}^{(i)}, z^{(i)})\}_{i=0}^{m}$, we have $z^{(i)} > 0$ for each i —particularly, we have $\boldsymbol{y}^{(0)} = \boldsymbol{b}_0$.

Under these conditions, we may safely assume that $\hat{f}(\boldsymbol{b}_0) > 0$. Due to the continuity property, there exists a neighborhood $B_\infty(\boldsymbol{b}_0, R)$ such that $\hat{f}(\boldsymbol{y}) > 0$ provided that $\boldsymbol{y} \in B_\infty(\boldsymbol{b}_0, R)$. Then, using bisectional detection, we may find a vector $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^{\mathrm{T}}$ and it guarantees $\hat{f}(\boldsymbol{y}) > 0$ if

$$-\epsilon_i \leq \boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{b}_0) \leq \epsilon_i \tag{4}$$

holds for each $1 \leq i \leq n$. We thus have $2n$ additional constraints in this case, we in what follows call them *feasibility constraints*.

3.3 Constraints Selection and Solving

There are two major approaches to dealing with this issue. The first is called *incremental constraint solving* under the condition that we can extract constraints from the trained model, and the second is called *bisectional detection* without constraints. The first algorithm is revealed in 1 , 2, 3 and 4. The second algorithm is revealed in 5.

Incremental Constraint Solving. To enhance clarity, we will explain this method in reverse order. With some fixed number N, starting from $\boldsymbol{y} = \boldsymbol{b}_0$, we may perform the optimization (w.r.t. fairness, robustness, etc) with not more than N constraints — each constraint may be either classification or feasibility, and suppose that we get an optimal value at $\boldsymbol{y} = \boldsymbol{b}$.

Then, for each $\boldsymbol{x} \in \mathcal{S}$ with $\mathcal{N}(\boldsymbol{x}) = i$, we get $k-1$ linear constraints for classification. As a result, we will in total have $O(\#\mathcal{S} \times k)$ classification constraints —remember that k is the number of all possible classes. This is in general not feasible for $\#\mathcal{S}$ is usually a large dataset, therefore, we have to reduce the amount of constraints.

Then, we need to examine if (2) holds, and let $\boldsymbol{b}_1 = \boldsymbol{b}$ in this situation. Otherwise, let

$$(\tilde{i}, \tilde{j}, \tilde{\boldsymbol{x}}) = \arg\min_{(i,j,\boldsymbol{x})} \{\mathcal{D}_{\boldsymbol{x}}^{i,j}(\boldsymbol{b}) \mid i, j < k, \boldsymbol{x} \in \mathcal{S}, \mathcal{N}_{\boldsymbol{b}_0}(\boldsymbol{x}) = i, \mathcal{D}_{\boldsymbol{x}}^{i,j}(\boldsymbol{b}) < 0\}$$

and we add $\mathcal{D}_{\tilde{\boldsymbol{x}}}^{\tilde{i},\tilde{j}}(\boldsymbol{b}) \geq 0$ into the constraint set then do the optimization again. Intuitively, for a sample $\boldsymbol{x}$ with $\mathcal{N}_{\boldsymbol{b}_0}(\boldsymbol{x}) = i$, the condition $\mathcal{D}_{\boldsymbol{x}}^{i,j}(\boldsymbol{b}) < 0$ for some j indicates a mis-classification of $\boldsymbol{x}$ with $\boldsymbol{y} = \boldsymbol{b}$. In this case, we call $\mathrm{abs}(\mathcal{D}_{\boldsymbol{x}}^{i,j}(\boldsymbol{b}))$ the *mis-classification degree* of $\boldsymbol{x}$ w.r.t. $\boldsymbol{b}$. Similarly, we need also to guarantee that (4) holds for every $i \leq n$. If it is not the case, let

$$\tilde{i} = \arg\max_i \left(\{\boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}_0) - \epsilon_i \mid \boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}_0) > 0\} \cup \{\boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}_0) + \epsilon_i \mid \boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}_0) < 0\}\right)$$

and we pick $\boldsymbol{e}_{\tilde{i}}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}_0) < \epsilon_{\tilde{i}}$ (resp. $-\epsilon_{\tilde{i}} < \boldsymbol{e}_{\tilde{i}}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}_0)$) into the constraint set if $\boldsymbol{e}_{\tilde{i}}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}_0) > 0$ (resp. $\boldsymbol{e}_{\tilde{i}}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}_0) < 0$).

Algorithm 1 Incremental Constraint Solving($\mathcal{N}$, $\boldsymbol{y}$, φ)

Input:

$\mathcal{N}$: A neural network $\mathcal{N}$ to be repaired.

$\boldsymbol{y}$: A collection of parameters to be repaired, which contains the position information of the parameters to be repaired.

φ: The high-level property to be repaired.

Output:

$\mathcal{N}'$: A repaired neural network, whose parameter values have been optimized.

/* Get a quadratic form $\frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{y} + \boldsymbol{p}^{\mathrm{T}}\boldsymbol{y} + c$, where $\boldsymbol{y}$ represents the value of the parameter to be repaired. */

1: $(\boldsymbol{Q}, \boldsymbol{p}, c)$ = QuadraticFitting($\mathcal{N}$, $\boldsymbol{y}$, φ)

/* Establish the constraints $\boldsymbol{A}\boldsymbol{y} \leq \boldsymbol{d}$ subject to the optimization. */

2: $(\boldsymbol{A}, \boldsymbol{d})$ = ExtractConstraints($\mathcal{N}$, $\boldsymbol{y}$)

3: Calculate $\mathcal{D}_x^{(i,j)}(\boldsymbol{y})$

/* Select the strongest constraints based on $\mathcal{D}_x^{(i,j)}(\boldsymbol{y})$. */

4: $(\boldsymbol{A}^-, \boldsymbol{d}^-)$ = SelectConstraints($\boldsymbol{A}$, $\boldsymbol{d}$)

/* The optimal solution $\boldsymbol{p}$ is obtained by quadratic optimization, where $\boldsymbol{b}$ represents the value of the repaired parameter. */

5: $\boldsymbol{b}$ = Optimize($\boldsymbol{Q}$, $\boldsymbol{p}$, c, $\boldsymbol{A}^-$, $\boldsymbol{d}^-$)

/* Modify the value of the parameter in $\mathcal{N}$ to get $\mathcal{N}'$.*/

6: $\mathcal{N}'$ = Repair($\mathcal{N}$, $\boldsymbol{y}$, $\boldsymbol{b}$)

7: return $\mathcal{N}'$

Bisectional Detection. Starting from $\boldsymbol{y} = \boldsymbol{b}_0$, suppose that the goal of optimization is to minimize $f_{\mathcal{N}}$ and we obtain the optimal point $\boldsymbol{y} = \boldsymbol{b}$ without any constraint. Since it may violate the classification and/or feasibility constraints, we intend to find some feasible point within the segment between $\boldsymbol{b}_0$ and $\boldsymbol{b}$. Nevertheless, this approach can be applied only when the following conditions are stated:

(1) The goal $f_{\mathcal{N}}$ is convex, i.o.w., $f_{\mathcal{N}} = \frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{y} + \boldsymbol{p}^{\mathrm{T}}\boldsymbol{y} + c$ for some $\boldsymbol{Q} \succeq \boldsymbol{0}$.

(2) $\mathcal{N}$ is locally robust at $\boldsymbol{b}_0$ w.r.t. $\mathcal{S}$, namely, there is some $r > 0$ such that $\mathcal{N}_{\boldsymbol{b}_0}(\boldsymbol{x}) = \mathcal{N}_{\boldsymbol{b}'}(\boldsymbol{x})$ holds for every $\boldsymbol{x} \in \mathcal{S}$ provided that $\boldsymbol{b}' \in \mathcal{B}(\boldsymbol{b}_0, r)$.

Similarly, when $\boldsymbol{Q}$ is indefinite, we adopt the same method described above to make the goal $f_{\mathcal{N}}$ convex.

Therefore, one may perform a bisectional detection to find some $\boldsymbol{b}_1 = \boldsymbol{b}_0 + \lambda(\boldsymbol{b} - \boldsymbol{b}_0)$ with $\lambda \in (0, 1]$, such that $\boldsymbol{b}_1$ is feasible, and in addition we have $f_{\mathcal{N}}\big|_{\boldsymbol{y}=\boldsymbol{b}_1} \leq f_{\mathcal{N}}\big|_{\boldsymbol{y}=\boldsymbol{b}_0}$ since $f_{\mathcal{N}}$ is convex.

Compositional Use. Similarly, starting from $\boldsymbol{b}_1$, we may alternatively use the above two approaches to obtain a set of new parameters $\boldsymbol{b}_2$ —of course, when bisectional detection is applied, two prerequisites should be fulfilled. Remind that we need recompute the goal to be optimized, the constraints with $\boldsymbol{b}_1$ in place of $\boldsymbol{b}_0$. Thus, a sequence of parameters $\{\boldsymbol{b}_i\}_i$ is obtained, and we can stop the process with the criterion $\|\boldsymbol{b}_t - \boldsymbol{b}_{t+1}\| < \epsilon$ for some t.

4 IMPLEMENTATION

Faulty Localization. In our approach, we leverage a state-of-the-art (SOTA) causality-based fault localization technique, as introduced in the work [41], to identify the neurons within a neural network that are primarily responsible for undesired behaviors. CARE is predicated on the Structural Causal Models (SCMs) framework, which provides a systematic way to represent the causal relationships between the components of a neural network.

Algorithm 2 QuadraticFitting($\mathcal{N}$, $\boldsymbol{y}$, φ)

Input:

$\mathcal{N}$: A neural network $\mathcal{N}$ to be repaired.

$\boldsymbol{y}$: A collection of parameters to be repaired, which contains the position information of the parameters to be repaired.

φ: The high-level property to be repaired.

Output:

$(\boldsymbol{Q}, \boldsymbol{p}, c)$: Parameters of quadratic forms.

/* $\boldsymbol{y_0}$ is the initial value of the parameter to be repaired in $\mathcal{N}$. */

/* Δ contains several perturbations imposed on the parameters. */

initialize Δ, $\boldsymbol{y_0}$

fitdata = Φ

$\Phi_0 = \emptyset$

for δ_i in Δ **do** :

/* When the value of the parameter to be repaired is $\boldsymbol{y_0} + \delta$, the compute function returns the score z of the neural network $\mathcal{N}$ on the verification property φ. */

    z_i = verify($\mathcal{N}$, $\boldsymbol{y}$, $\boldsymbol{y_0} + \delta_i$, φ)

/* Put the data needed for fitting into the list. */

    $\Phi_i \leftarrow (\boldsymbol{y_0} + \delta_i, z_i)$

    $\Phi = \Phi \cup \Phi_i$

end for

/* Call the fitting function to get the parameters of the quadratic form. */

$(\boldsymbol{Q}, \boldsymbol{p}, c)$ = fitting(Φ)

return $(\boldsymbol{Q}, \boldsymbol{p}, c)$

Algorithm 3 ExtractConstraints($\mathcal{N}$, $\boldsymbol{x}$, $\boldsymbol{y}$)

Input:

$\mathcal{N}$: A neural network $\mathcal{N}$ to be repaired.

$\boldsymbol{x}$: The input of neural network $\mathcal{N}$.

$\boldsymbol{y}$: A collection of parameters to be repaired, which contains the position information of the parameters to be repaired.

Output:

$(\boldsymbol{A}, \boldsymbol{d})$: Classification constraints.

Starting with the input $\boldsymbol{x}$, the desirable classification of $\mathcal{N}(\boldsymbol{x})$ is $\boldsymbol{i}$

for every label $\boldsymbol{j} \neq \boldsymbol{i}$ **do**:

/* The condition $\mathcal{D}_x^{i,j}(\boldsymbol{y}) < 0$ for some j indicates a mis-classification of $\boldsymbol{x}$. */

    Calculate $\mathcal{D}_x^{(i,j)}(\boldsymbol{y})$

    Get positive classification constraints $\boldsymbol{A_p y} \leq \boldsymbol{d_p}$ from $\mathcal{D}_x^{(i,j)}(\boldsymbol{y}) \geq 0$

    Get negative classification constraints $\boldsymbol{A_n y} \leq \boldsymbol{d_n}$ from $\mathcal{D}_x^{(i,j)}(\boldsymbol{y}) < 0$

end for

$(\boldsymbol{A}, \boldsymbol{d}) = (\boldsymbol{A_p} \cup \boldsymbol{A_n}, \boldsymbol{d_p} \cup \boldsymbol{d_n})$

return $(\boldsymbol{A}, \boldsymbol{d})$

Algorithm 4 SelectConstraints($\boldsymbol{A}$, $\boldsymbol{d}$)

Input:

$\boldsymbol{x}$: The input of neural network $\mathcal{N}$.

$\boldsymbol{y}$: A collection of parameters to be repaired, which contains the position information of the parameters to be repaired.

$\mathcal{D}_x^{(i,j)}(\boldsymbol{y})$: Mis-classification degree.

$(\boldsymbol{A}, \boldsymbol{d})$: Classification constraints.

Output:

$(\boldsymbol{A}^-, \boldsymbol{d}^-)$: Selected classification constraints.

/* Select negative constraints with higher mis-classification degree. */

$(\tilde{i}, \tilde{j}, \tilde{\boldsymbol{x}}) = \arg\min_{(i,j,\boldsymbol{x})} \{\mathcal{D}_{\boldsymbol{x}}^{i,j}(\boldsymbol{y}) \mid i, j < k, \boldsymbol{x} \in \mathcal{S}, \mathcal{N}_{b_0}(\boldsymbol{x}) = i, \mathcal{D}_{\boldsymbol{x}}^{i,j}(\boldsymbol{y}) < 0\}$

Get negative classification constraints $\boldsymbol{A_n}^- \boldsymbol{y} \leq \boldsymbol{d_n}^-$ from $\mathcal{D}_x^{(\tilde{i},\tilde{j})}(\boldsymbol{y}) < 0$

/* Select positive constraints with lower $\mathcal{D}_x^{(i,j)}(\boldsymbol{y})$. */

$(\tilde{i}, \tilde{j}, \tilde{\boldsymbol{x}}) = \arg\min_{(i,j,\boldsymbol{x})} \{\mathcal{D}_{\boldsymbol{x}}^{i,j}(\boldsymbol{y}) \mid i, j < k, \boldsymbol{x} \in \mathcal{S}, \mathcal{N}_{b_0}(\boldsymbol{x}) = i, \mathcal{D}_{\boldsymbol{x}}^{i,j}(\boldsymbol{y}) \geq 0\}$

Get positive classification constraints $\boldsymbol{A_p}^- \boldsymbol{y} \leq \boldsymbol{d_p}^-$ from $\mathcal{D}_x^{(\tilde{i},\tilde{j})}(\boldsymbol{y}) \geq 0$

/* Merge classification constraints. */

$(\boldsymbol{A}^-, \boldsymbol{d}^-) = (\boldsymbol{A_p}^- \cup \boldsymbol{A_n}^-, \boldsymbol{d_p}^- \cup \boldsymbol{d_n}^-)$

/* Add feasibility constraints. */

if not $-\epsilon_i \leq \boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{b}_0) \leq \epsilon_i$ **then**

  $\tilde{i} = \arg\max_i \left(\{\boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{b}_0) - \epsilon_i \mid \boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{b}_0) > 0\} \cup \{\boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{b}_0) + \epsilon_i \mid \boldsymbol{e}_i^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{b}_0) < 0\}\right)$

  pick $\boldsymbol{e}_{\tilde{i}}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{b}_0) < \epsilon_{\tilde{i}}$ (resp. $-\epsilon_{\tilde{i}} < \boldsymbol{e}_{\tilde{i}}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{b}_0)$) into the constraint set $(\boldsymbol{A}^-, \boldsymbol{d}^-)$

end if

return $(\boldsymbol{A}^-, \boldsymbol{d}^-)$

Our application of CARE begins with modeling the neural network as an SCM, where each neuron is considered an endogenous variable influenced by both exogenous variables and other neurons within the network. By doing so, we can quantify the causal effect of each neuron on the output, which is critical for identifying the sources of defects. The causal attribution of a neuron is calculated by intervening on its value and observing the change in the network's undesirable behavior, measured as the deviation from the desired property. This interventional expectation is computed by sampling inputs from their distribution while repairing the neuron's value, and then averaging the model's behavior over these samples.

In our work, we apply this fault localization method to pinpoint the neurons that contribute significantly to the violation of specific properties, such as fairness or robustness. Once these neurons are identified, we select the parameters corresponding to these neurons as variables, while fixing the parameters corresponding to the other neurons.

Properties Calculation. AutoRIC primarily addresses quantifiable properties such as fairness and robustness. The current implementation of AutoRIC framework is limited to measuring these two properties. Future work aims to extend the framework to include additional properties and develop a universal specification language to characterize network properties. However, designing certain specification presents significant challenges at this stage.

First of all, we need to make some neccessary preparations before calculating fairness. We first classify the datasets to different groups based on the value of protected features. Then we calculate the specific values using the difference of distribution of the favorable label to measure fairness.

Algorithm 5 Bisectional Detection($\mathcal{N}$, $\boldsymbol{y}$, φ)

Input:
$\mathcal{N}$: A neural network $\mathcal{N}$ to be repaired.
$\boldsymbol{y}$: A collection of parameters to be repaired, which contains the position information of the parameters to be repaired.
φ: The high-level property to be repaired.
hasConstraint: A boolean variable indicating whether constraints can be extracted from the network.
Output:
$\mathcal{N}'$: A repaired neural network, whose parameter values have been optimized.

Assumptions:
  The goal of optimization is to minimize $f_{\mathcal{N}}$.
  The goal $f_{\mathcal{N}}$ is convex.
  $\mathcal{N}$ is locally robust at $\boldsymbol{b}_0$.
if *hasConstraint* **then**:
  /* Call another algorithm. */
    Incremental Constraint Solving($\mathcal{N}$, $\boldsymbol{y}$, φ)
else
    Starting from $\boldsymbol{y} = \boldsymbol{b_0}$, suppose that we obtain the optimal point $\boldsymbol{y} = \boldsymbol{b_1}$ without any constraint.
    while optimal point $\boldsymbol{b}$ violate the classification and/or feasibility constraints **do**:
      $\boldsymbol{b_i} = \boldsymbol{b_{i-1}} + \lambda(\boldsymbol{b_{i-2}} - \boldsymbol{b_{i-1}})$ with $\lambda \in (0, 1]$, $i = 2, 3, \ldots$
    end while
end if
$\mathcal{N}'$ = Repair($\mathcal{N}$, $\boldsymbol{y}$, $\boldsymbol{b_i}$)
return $\mathcal{N}'$

However, the proportion of the favorable prediction is not equal to the distribution of the favorable prediction considering that the dataset cannot fully reflect the real-world. To tackle this issue, we perform a uniform-sampling just on protected (independent) features, and then we obtain the distribution of other features. Finally, we combine such two types of features and the corresponding sampling labels. With 10000 repetitions of sampling, we obtain data with the same amount, and we regard them as the input of trained neural networks to calculate the difference of proportion of specific protected feature. i.e. we gain the fairness of specific protected feature. It is worth noting that CARE and the other state-of-the-art approaches repair fairness based on original datasets.

Meanwhile, we use robustness radius to measure robustness. Since the typical form of a convex optimization problem is to minimize a convex function subject to a set of convex constraints. We regard the the reciprocal of the robustness radius as the optimization objective. Then we employ state-of-the-art tools for verifying the robustness of neural networks to calculate robustness radius. In AutoRIC, we integrate DeepZ to calculate robustness radius. DeepZ [35] is a robust neural network verification tool based on Zonotope-based abstraction. It leverages abstract interpretation techniques to efficiently handle input uncertainties and provide formal proofs of a network's robustness. DeepZ is particularly suitable for various neural network architectures, including deep neural networks and convolutional neural networks. By using Zonotope-based abstraction, DeepZ can perform robustness verification with lower computational costs, making it a powerful tool for ensuring the reliability of neural network models under specific input perturbations.

Sampling and Fitting. In the following, we introduce concrete sampling procedures to mapping the faulty parameters and properties. Given a trained network and faulty parameters, we randomly sample new parameters within the range of parameter values. Then we gain a set of sampled parameter values, which are used to modify the network accordingly. We calculate certain property values for these modified networks based on the aforementioned computational method. Meanwhile, we record the mapping relationship between the sampled parameter values and the corresponding property values.

After determining the coefficients such as c, $\boldsymbol{p}$ and $\boldsymbol{Q}$ through fitting, we identify the neurons with significant contributions in the hidden layers. For each neuron, given the corresponding parameters, the dimensionality of $\boldsymbol{Q}$ is determined by the number of parameters. Based on the fitting results, we approximate the function $f(\boldsymbol{y})$ as follows:

$$f(\boldsymbol{y}) \approx \hat{f}(\boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{y} + \boldsymbol{p}^{\mathrm{T}}\boldsymbol{y} + c$$

Optimization with Constraints. Applying the localization method of CARE, we localize the initial faulty parameters, and then select parameters from a single layer near to the output. We categorize the input data into two classes, one with accurate prediction and one with erroneous prediction.

For the network whose activation function is differentiable (or, piece-wise differentiable), the behavior within each linear region of the network is linear and can be approximated. Based on the linearization technique of PRDNN, we linearize the certain single layer of network by calculating the Jacobian at specific points, i.e. faulty neurons. This linear approximation is precise because these functions are differentiable almost everywhere within their domain. Computing the Jacobian matrix at a specific point involves calculating the derivative of the network's output with respect to the inputs (or weight vector) at that particular layer and input point. For certain faulty neuron, we can obtain the linear relationship between its output and its input. This is also the reason why incremental constraint solving only supports differentiable networks.

For the inputs which are classified correctly, we linearize the faulty neurons and gain linear relationships between output and input of neurons, which are extracted as positive constraints. Conversely, for misclassified inputs, we extract negative constraints. On the one hand, quantities of constraints may all result in no solution. Therefore, we select top percentage positive constraints sorted in descending order and negative constraints sorted in ascending order. The specific percentage needs to be determined based on the dataset and the trained network, and this will be elaborated on in the experimental evaluation section. Subsequently, feasibility constraints are incorporated as necessary based on the specific context. Ultimately, we solve the quadratic programming problem to gain the optimal of faulty parameters.

Optimization without Constraints. For networks that are non-differentiable, we currently cannot extract constraints. Therefore, we attempted to optimize the parameters to be repaired using the bisection detection method. Starting from $\boldsymbol{y} = \boldsymbol{b_0}$, we randomly determine the value $\boldsymbol{y} = \boldsymbol{b}$ of faulty parameters within range at first. Since it may violate the classification and/or feasibility constraints, we intend to find some feasible point within the segment between the $\boldsymbol{b_0}$ and $\boldsymbol{b}$. Then we perform bisection detection method to repair the faulty neurons. The details is shown in section 3.

5 EXPERIMENTAL EVALUATION

We have implemented the approach based on PyTorch and Scikit-learn. The source code and benchmark are available at https://github.com/Anonymous89000/AutoRIC. All experiments are conducted on a machine with Intel Core i7 3.0GHz CPU, 32GB system memory and 1 NVIDIA GTX 1660Ti GPU.

Table 1. Benchmark datasets for fairness repair

Name	P.Feature	#Features	Favorable label	Size
Census	gender, age, race	14	1(income>50K)	32, 561
Bank	age	17	1(good credit)	45,211
Jigsaw	race	\	0(toxic)	313,000

5.1 Experimental Setup

We apply AutoRIC to two repair tasks: 1) fairness repairing tasks and 2) robustness repairing tasks. We adopt retraining, CARE [41] and other related work as baselines for comparison.

Fairness repairing tasks. In the following experiments, we focus on group fairness. Group fairness examines whether subpopulations with different sensitive attributes are treated with equity by the learned model [13]. The task aims to improve the fairness of the training model by making minor modifications to the parameters, while ensuring minimal decrease on the model's accuracy.

We compare AutoRIC with state-of-the-art (SOTA) method CARE and PVNN on fairness repairing tasks. CARE is a systematic approach to repair properties such as fairness and safety, and performs sufficient experiments on kinds of datasets. The experimental design is comprehensive and the effects are pronounced. PVNN is an approach to formally verify neural networks against fairness. CARE and PVNN are all implemented on the SOCRATES [33] platform, a unified platform for neural network analysis.

For fairness repair, we adopt three open-source datasets to be our experiment subjects, which have been used in previous work[40] on fairness testing. The three datasets that are commonly used in machine learning model fairness testing. The details of the datasets are depicted as Table 1. In that table, the column P.Feature lists protected features of that dataset, where trained neural networks should function fairly on different input data with different value of the protected feature. Intuitively, fairness issues arise when a model makes different decisions for instances that differ only by sensitive attributes, such as age, race, or gender.

- Census income [22] consists of 32, 561 instances with 14 features, among which gender, age and race are protected features. The dataset is used to predict whether an adult's income exceeds $50K per year.
- Bank Marketing [31] consists of 45, 211 instances with 17 features, among which age is the protected feature. The dataset is used to predict if the client will subscribe a term deposit.
- Jigsaw [7] consists of 313, 000 text comments, among which race is the protected feature. The dataset is aimed to classify whether the text comments are toxic or non-toxic.

Following the existing work [2, 14, 39, 41, 45, 51], we train three 6-layer feed-forward neural networks (FFNNs) on the first two datasets. Then we calculate fairness of the corresponding feed-forward networks based on specific protected features. Then we repair unfair networks for each protected feature. The details of models are depicted in Table 2. It shows the architecture, the amount of parameters and initial accuracy of each model. For Jigsaw, we train classifier using 3 long short term memory (LSTM) layers and 1 dense layers. Following previous work, we use the state-of-the-art embedding tool Glove [32] and adopt 50-dimension word vectors to encode text comments. The accuracy of the classifier is 0.93.

Robustness repairing tasks. Robustness repair tasks involve identifying and correcting vulnerabilities in a neural network to ensure consistent and reliable performance under various conditions. These tasks include fault localization, followed by the application of techniques to improve the

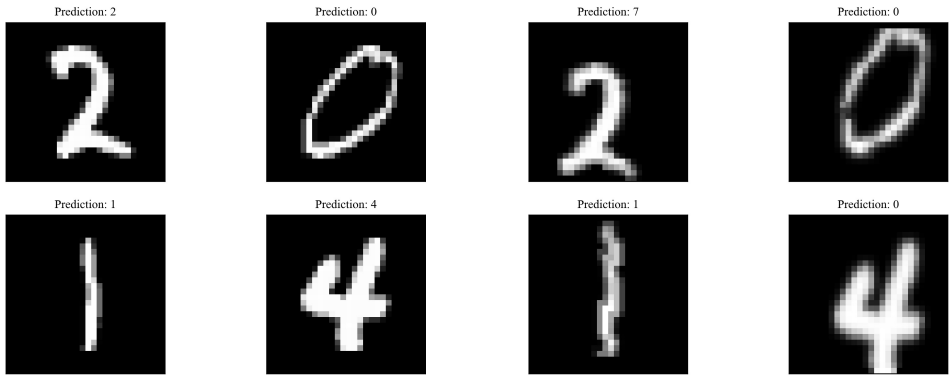
Table 2. Experimental Models

Model	Dataset	P.Feature	Architecture	#Parameters	Accuracy
M1	Census	gender, age, race	6-layer FFNN	3,750	0.84
M2	Bank	age	6-layer FFNN	3,878	0.892
M3	Jigsaw	race	3-layer LSTM + 1-layer Linear	4,162	0.93
M4	MNIST	\	2-layer Conv + 3-layer Linear	358,186	0.99

network's resistance to adversarial attacks and perturbations. By refining the network parameters and enhancing its robustness, we aim to maintain the network's accuracy and stability, even in the presence of unexpected inputs or challenging environments. The robustness radius is an important metric for assessing the robustness of a neural network; the larger the robustness radius, the stronger the network's robustness.

For robustness repair, we adopt MNIST [25] as experimental dataset, which is a classic dataset for verifying the robustness of neural networks. In detail, MNIST consists of 70, 000 handwritten digits as images, among which 60, 000 images are used for training and 10, 000 images are used for testing. The images are gray-scale and centered, with 28×28 pixels. MNIST need to be classified into one of 10 digits.

Based on PyTorch, we train classifier on MNIST using 2 convolutional layers and 3 dense layers with 0.99 accuracy. This convolutional neural network has significant effects on classifying MNIST. Notably, even with noise, the classification accuracy of trained MNIST model may decrease significantly. To illustrate, as is shown in Figure 2(a) and 2(b), after adding effective perturbations such as random rotation, resize, affine transformations and sharpness adjustments, the classification accuracy decrease from 100% to 50%.



(a) Prediction of MNIST without perturbation. (b) Prediction of MNIST with perturbation.

Fig. 2. Prediction diagram of MNIST

5.2 Research Questions

We report experimental results to answer the following research questions, demonstrating AutoRIC's effectiveness and efficiency. We aim to answer the following four research questions:

- *RQ1:* Is AutoRIC successful in neural network repairing?

- *RQ2:* Are the constraints necessary?
- *RQ3:* How about the effectiveness and efficiency of AutoRIC?
- *RQ4:* Why choose Quadratic Programming?

RQ1: Is AutoRIC successful in neural network repairing*?*

We aim to construct a universal repair approach, which can repair various kinds of properties of various kinds of neural networks. To answer this question, we conduct experiments that explore two aspects.

Fairness Repairing. Our approach works on any network whose activation function is differentiable (or, piece-wise differentiable). Using AutoRIC, we first repair fairness of FFNN and LSTM on aforementioned datasets. We describe the details of our experiments in the following.

Following the previously mentioned procedures for fault localization, fairness calculation, and sampling, we establish the mapping relationship between the faulty parameters and fairness. Meanwhile, we determine the coefficients like c, $\boldsymbol{p}$ and $\boldsymbol{Q}$ after fitting. Take the fairness of feature gender of census dataset for example, the faulty neurons which have outstanding contributions are the 7^{rd} neuron, 11^{rd} neuron and 12^{rd} neuron at 3^{th} hidden layer. Each neuron has 8 parameters, the dimension of $\boldsymbol{Q}$ is 24×24. Based on the result of fitting, we get the approximation of $f(\boldsymbol{y}_{24,1})$.

$$f(\boldsymbol{y}_{24,1}) \approx \hat{f}(\boldsymbol{y}_{24,1}) = \frac{1}{2}\boldsymbol{y}_{24,1}^{\mathrm{T}}\boldsymbol{Q}_{24,24}\boldsymbol{y}_{24,1} + \boldsymbol{p}_{24,1}^{\mathrm{T}}\boldsymbol{y}_{24,1} + c$$

Then we extract and select the classification and feasibility constraints and perform quadratic programming with constraints. It is worth noting that we extract constraints from testing dataset. For example, the testing dataset of census consist of 7543 instances of data and the testing dataset of census consist of 11303 instances of data. According to the optimization results, we obtain the optimal values of the parameters to be repaired.

Table 3 reveals the results of repairing with constraints. The table provides the parameters after localization, positive constraints sorted in descending order, negative constraints sorted in ascending order, in addition to the fairness increase percentage and accuracy drop percentage. Fairness increase means the training model exhibits less bias when when facing different groups, treating all categories more fairly. Accuracy drop means a decline in the network's performance on classification tasks. Taking the age feature of bank dataset as example, we choose top 0.04% positive constraints sorted in descending order and top 0.4% negative constraints sorted in ascending order. The faulty neurons which have the largest contributions are the 7^{rd} neuron, 11^{rd} neuron and 12^{rd} neuron at 3^{th} hidden layer. AutoRIC successfully repairs M2, with 100% fairness improvement and the accuracy drops slightly(1%). For census dataset, we select less percentage of both positive and negative constraints. For example, we choose top 0.2% positive constraints and top 0.2% negative constraints for protected feature—gender. Therefore, it is essential to select appropriate percentage of constraints for specific dataset.

Table 4 concludes the fairness repairing results, which includes model, protected feature, fairness before/after repairing, accuracy before/after repairing, and the percentage of fairness increase and accuracy drop. Table 4 demonstrates that distinct datasets have distinct fairness concerns. Bearing in mind that fairness is the measure of evenness of distribution, the lower the value the more equitable the networks tend to be. For M1, the fairness of gender is more outstanding with 0.138. AutoRIC has an average 51.17% and 3.77% fairness increase and accuracy decrease. AutoRIC is able to repair the fairness of M2 to a near-zero level with slight accuracy drop (1%). Meanwhile, for M3 with LSTM cells, AutoRIC increases fairness by 66.67%, with a mere 2.15% decrease in accuracy.

Robustness Repairing. To verify the generality of our framework, we measure robustness by calculating the robustness radius and apply it to our framework. For robustness repair tasks, we use DeepZ [35] to verify the robustness of neural networks. This verification tool can process quickly

Table 3. Details of Repairing

Model_Feature	Param. localization	Pos-constraints(d)	Neg-constraints(a)
Census_gender	(3,7), (3,11), (3,12)	0.20%	0.20%
Census_age	(3,2), (3,7), (3,10)	0.40%	0.10%
Census_race	(3,3), (3,7), (3,10)	0.60%	0.20%
Bank_age	(3,7), (3,11), (3,12)	0.04%	0.40%

* Pos-constraints(d), Neg-constraints(a) is short for positive constraints sorted in descending order, negative constraints sorted in ascending order.

Table 4. Fairness Repair with Constraints

Model	P.Feature	Fair-B	Fair-A	Acc-B	Acc-A	Fair-Increase(%)	Acc-Decrease(%)
M1	gender	0.138	0.069	0.840	0.814	**50.22**	3.01
M1	age	0.076	0.0441	0.840	0.829	**41.97**	1.32
M1	race	0.076	0.0299	0.840	0.803	**60.66**	4.40
M2	age	0.048	0.000	0.892	0.883	**100.00**	1.00
M3	race	0.060	0.020	0.930	0.910	**66.67**	2.15

* P.Feature, Fair-B, Fair-A, Acc-B, Acc-A, Fair-Increase, Acc-Decrease is short for Protected Feature, Fairness Before, Fairness After, Accuracy Before, Accuracy After, Fairness Increase, Accuracy Decrease.

Table 5. Robustness Repair

Model	Robust-B	Robust-A	Acc-B	Acc-A	Robust-Increase(%)	Acc-Decrease(%)
M4	0.126	0.140	0.990	0.980	**15.87**	1.01

* Robust-B, Robust-A, Robust-Increase is short for Robustness Before, Robustness After, Robustness Increase.

when the answer is accurate, or it may take longer. The robustness radius of M4 was improved by 15.87% (from 0.126 to 0.140) with a 1.01% decrease in accuracy.

In summary, AutoRIC successfully addresses neural network issues and significantly increases fairness by over 50% on average. The robustness radius has a 15.87% increase. However, further experimentation reveals an inverse relationship between neural network properties and classification accuracy when constraints are adjusted. When the fairness reaches a specific percentage, there will be a significant drop in accuracy. Since fairness and accuracy do not always align, it is challenging to maintain a high degree of accuracy while ensuring fairness. Therefore, we select a balance between improving fairness and minimizing the drop in accuracy within a set range, ultimately finding the optimal balance point.

Answer to RQ1 AutoRIC effectively transfers the neural network repairing to an optimization problem and successfully repairs quantitative properties such as fairness and robustness. Meanwhile, further experimentation reveals an inverse relationship between neural network properties and classification accuracy when constraints are adjusted. It is essential to select a balance between repairing and maintaining accuracy.

RQ2: Are the constraints necessary?

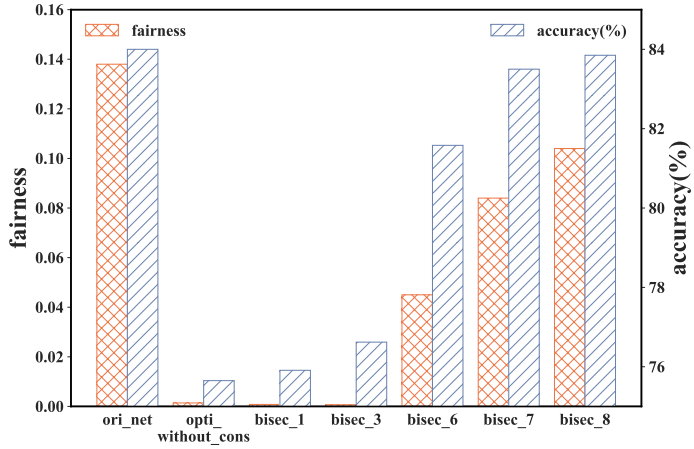


Fig. 3. Optimization results without constraints of M1_Gender

Note that we currently can not extract the constraints from all kinds of trained neural networks since the activation function which is not differentiable. Therefore we propose another repairing method without constraints. In practice, we combine the optimization without constraints and bisectional detection method to repair the trained networks since unconstrained optimization may result in a large drop of accuracy. The frequency of bisectional detection depends on the results of fairness and accuracy. Performance of this method is demonstrated to be linked to the dataset and model through experimental results.

Figure 3 illustrates the results of the experimental results of *unconstrained optimization with bisectional detection* (UOBD, for short) for M1 w.r.t. the protected feature gender. As it demonstrates, fewer bisectional search after optimization will produce an alarming decrease in fairness and accuracy. Such a search strategy works most effectively at the setting of time = 7.

Figure 4 illustrates the results of the experimental results of UOBD for M1 w.r.t. the protected feature age. Figure 4 follows the same trend as Figure 3 regarding fairness and accuracy. Nevertheless, when split three times, the fairness and accuracy is comparable to the initial network. Evidently, this method is ineffective for this feature and model.

Figure 5 reveals the results of the experimental details of UOBD for M2 w.r.t. the protected feature age. As a whole, the accuracy and fairness all diminish as further bisections are done. By dividing three times, the outcome gives the best outcome. The fairness decreases to zero with a slight drop in accuracy(1%). This method is evidently successful for this dataset and model.

Answer to RQ2 As a result, optimization without constraints is not stable in its performance and output, therefore, imposing constraints when performing repairing is necessary.

RQ3: How about the effectiveness and efficiency of AutoRIC?

For the effectiveness of AutoRIC, we choose Retraining, CARE and PVNN as the baseline. Table 6 illustrates the results, including fairness improvement percentage, accuracy decrease percentage and cost of time.

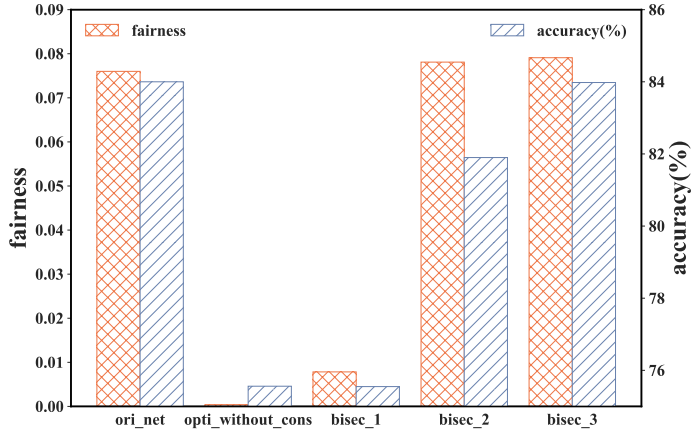


Fig. 4. Optimization results without constraints of M1_Age

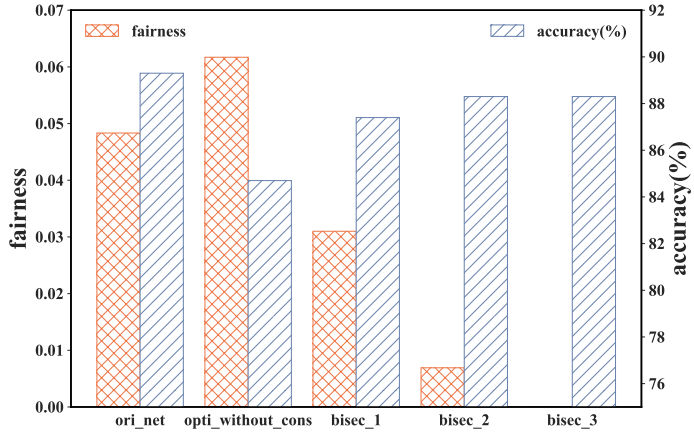


Fig. 5. Optimization results without constraints of M2_Age

Compared with Retraining, AutoRIC achieves a higher percentage increase of about 42.56% in fairness for all features compared to retraining, indicating that AutoRIC is more effective in improving model fairness. Meanwhile, AutoRIC results in a lower percentage decrease of about 1.86% in accuracy than retraining, suggesting that AutoRIC is better at maintaining the original accuracy of the model.

Compared with CARE, AutoRIC has the advantage in terms of fairness improvement and time consumption. However, for M1, AutoRIC has a higher decrease in accuracy of about 1.1% compared to CARE. We attribute this to the inability to achieve the same high initial accuracy as CARE. For

Table 6. Comparison with Retraining, CARE and PVNN

Model	Feature	Tech.	Fair-Increase(%)	Acc-Decrease(%)	Time(s)
M1	gender	AutoRIC	**50.22**	**3.01**	**79**
		Retraining	26.81	4.18	1
		CARE	46.15	2.27	240
	age	AutoRIC	**41.97**	**1.32**	**112**
		Retraining	5.70	1.43	1
		CARE	30.50	2.27	276
	race	AutoRIC	**60.66**	**4.40**	**139**
		Retraining	13.16	5.42	1
		CARE	73.40	3.40	314
M2	age	AutoRIC	**100.00**	**1.00**	**69**
		Retraining	77.43	1.09	2
		CARE	94.85	1.09	435
M3	race	AutoRIC	**66.67**	**2.15**	**79**
		PVNN	21.96	0.70	1638

Table 7. Time cost of tasks

Model	Feature	Fitting(s)	Linearize(s)	Optimization(s)	Total(s)
M1	gender	54.10	0.07	24.76	78.93
M1	age	84.77	0.05	26.78	111.60
M1	race	113.81	0.04	25.57	139.42
M2	age	63.68	0.04	4.51	68.23
M3	race	74.11	—	4.65	78.76
M4	—	8192	—	0.23	2209.95

M2, AutoRIC performs best in both accuracy decrease and fairness improvement. For M3, results show that AutoRIC yields 3 times more fairness improvement than PVNN.

For the efficiency of AutoRIC, the majority of time cost comes from quadratic fitting and quadratic programming, which all take polynomial time in the number of faulty parameters. Concerning quadratic fitting, there are two steps —the first step is to generate discrete points used for fitting, i.e. a pair of fairness and faulty parameters, the second step is to perform quadratic fitting to gain the quadratic function. Followed by optimization with constraints or optimization without constraints and bisection. Table 6 reveals the total time comparison with CARE and PVNN. In total , our approach is around 2 and 7 times more efficient than CARE and PVNN in fairness tasks. Considering fairness improvement, accuracy decrease, and time efficiency, AutoRIC offers superior overall performance compared to others.

Table 7 provides the cost of time for performing fitting and UOBD, respectively. It has been experimentally evaluated that AutoRIC is efficient. It is worth mentioning that the fitting cost of M4 depends on the robustness verification by DeepZ. "— " indicates that the experimental setup lacks the protected feature or that a certain operation is not performed.

Table 8. Comparison of two optimization methods

Model_Feature	Method	Fair-Increase(%)	Acc-Decrease(%)	Time(s)
M1_gender	QP	**78.99**	**4.26**	**80**
	LP	72.46	6.49	55
M1_age	QP	**90.39**	**6.63**	**114**
	LP	86.89	8.61	88
M1_race	QP	**84.87**	**7.66**	**135**
	LP	84.64	8.55	110

Answer to RQ3 It is shown that AutoRIC is effective and efficient in repairing tasks. With a slight accuracy decrease, AutoRIC repairs the high-level properties with more increase than state-of-art methods. AutoRIC is around 2 and 7 times more efficient than CARE and PVNN in fairness tasks. Likewise, AutoRIC repairs robustness within a short time.

RQ4: Why choose Quadratic Programming? Neural networks can model complex, non-linear relationships between inputs and outputs. Therefore, it is difficult to convert the repair task into a linear programming problem. Quadratic programming (QP) performs better than linear programming (LP) to model this problem, and the following experimental results also illustrate this concept. In table 8, We compare the fairness repair effectiveness of two methods on M1. After selecting appropriate amounts of constraints, linear programming has greater accuracy drop with less fairness improvement. For the feature gender of M1, QP improves the fairness by 78.99% with 4.26% accuracy decrease while LP improves the fairness by 72.46% with 6.49% accuracy decrease. In total, M1 outperforms LP by 3.42% on average when considering fairness, while sacrificing 1.7% in accuracy. Considering fairness and accuracy can not achieve optimal results at the same time, there is a trade-off between fairness and accuracy. Therefore, LP is not an effective option for performing repair task. Futhermore, theoretical support for programming beyond quadratic programming has not been sufficient. So we choose to use QP in our repair framework. Compared to QP, LP method takes significantly less time for optimization, almost negligible, while the time consumption in other aspects is similar.

Answer to RQ4 When performing the same repairing task, QP results in a smaller decrease in accuracy than LP. Therefore, QP has the upper hand in repairing effectiveness.

6 DISCUSSION

Based on the aforementioned experiments, we observed a potential tradeoff between the network's properties and its performance. For instance, in the fairness repair task concerning the protected feature gender within the census dataset, we selected eight sets of constraints with varying proportions of positive and negative constraints. These serve as constraints to solve quadratic programming, resulting in eight different results. Figure 6 illustrates the concurrent trend of increase of fairness and decrease of accuracy. The experimental results exhibit a pattern where the percentage increase in network fairness and the percentage decrease in accuracy tend to show a roughly positive correlation. Certainly, not all repair tasks exhibit such a strong positive correlation. The general relationship between properties and network performance still requires further investigation. Other

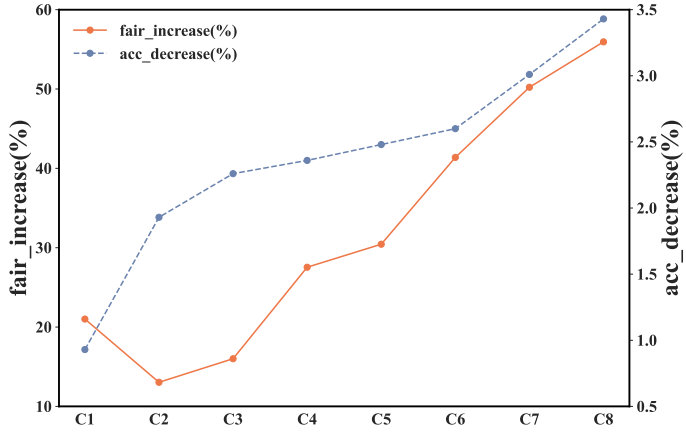


Fig. 6. The fairness increase trend and accuracy decrease trend of M1_Gender

related works such as FairNeuron [13] also observe that optimizing accuracy and fairness can be contradictory goals in training.

7 RELATED WORK

Currently, there are three mainstream repairing approaches, namely retraining, fine-tuning without fault localization and fine-tuning with fault localization. Towards the pre-trained neural networks, the retraining approach trains the neural networks again to satisfy certain properties. Fine-tuning focuses on repairing a specific subset of neural network parameters. Fault localization can identify such a subset that have a greater impact on misbehavior of neural networks. This procedure can make fine-tuning more targeted and purposeful. In the following, we introduce the related works mentioned above respectively.

With reaching analysis, Veritex [49] can repair unsafe neural networks that are used in safety-critical systems. It calculates the reachable domain of unsafe DNNs and during retraining adds a loss function term to optimize parameters. This approach can also repair faulty behavior of DNN Agent combined with deep reinforcement learning algorithms. The FairNeuron [13]and RUNNER [43] are both approaches aimed at enhancing the fairness of deep neural networks by addressing unfair neurons. FairNeuron is a method designed to enhance fairness in learning models by employing a multi-step diagnosis process that includes profiling and analysis to identify neurons that contribute to fairness. Its strengths lie in its comprehensive approach to understanding model fairness and its capability to selectively retrain on specific samples. However, FairNeuron presents certain disadvantages, such as high computational overhead due to its iterative diagnosis process and the complexity involved in hyperparameter tuning. RUNNER is an efficient and effective framework for enhancing fairness in machine learning models. It simplifies the diagnosis of unfair neurons through an importance-based criterion and stabilizes them via retraining with a tailored loss term. The advantages include reduced computational overhead, direct reduction of neuron discrimination, and robust generalization across different data types. However, the framework also has limitations, such as the necessity of a white-box model to access internal parameters and the reliance on a single hyperparameter, which requires careful calibration for optimal neuron

selection. Ethical Adversaries [9] framework integrates adversarial machine learning to improve fairness in predictive models. This framework employs a dual-adversary approach, utilizing evasion attacks to iteratively refine the model, with the objective of improving demographic parity and equal opportunity while maintaining utility. Key strengths of the framework include its innovative methodology, empirical validation, and utility preservation. However, it faces challenges such as computational complexity, risk of overfitting, generalizability concerns, reliance on surrogate models, and sensitivity to hyperparameter tuning.

REASSURE [12] is the first sound and complete repair framework for ReLU networks with strong theoretical guarantees. This technique proposes to use a patch function and it supports whole network repair. Unlike other repairing strategies, REASSURE changes the function space locally and guarantees the removal of the buggy behavior. Knowing that ReLU networks are piece-wise linear (PWL), this method constructs a patch network working on the linear space of faulty input. This method has both completeness and soundness guarantees. PRDNN [39] is a provable deep neural network repairing algorithm, which introduces a decoupled DNN based on the original one, then it reduces the repair problem to a linear programming problem. Followed by APRNN, it expands the PRDNN to V-polytope provable repair and can modify multiple layers of the DNN with polynomial time. However, this approach aims to improve the accuracy of models, e.g. MNIST. Our work pays attention on the properties of trained models. MDNN [15] is a method making provably minimal modifications of Deep Neural Networks without retraining and less unexpected effects on behavior. Combined with newest verification techniques, this approach proposes a modified technique with specific demand. But this approach is shown to be NP-complete and time-consuming. Editable Fine-Tuning [36] proposes a training technique which can quickly edit the trained model. However, this method can not guarantee the soundness and completeness. LocalRepair [30] offers general applicability to various neural networks, emphasizes layer-wise repairs using Mixed-Integer Quadratic Programming (MIQP), and maintains model accuracy by minimizing modifications. Nonetheless, the approach confronts scalability due to computational demands, susceptibility to overfitting, and challenges in reflecting overall network behavior. The success of this method is also contingent upon the linearity of predicate representation for constraints. Counterexample-guided-Repair [5] explores the robust optimization perspective for counterexample-guided neural network repair, highlighting its potential for safety-critical applications. The method's strengths are highlighted, including its proven termination for specific models and the development of an efficient repair algorithm based on QP. However, it also acknowledges limitations, including the lack of termination guarantee in general settings and the theoretical disadvantage of using early-exit verifiers.

NNRepair [46] is a constraint-based technique for repairing neural network classifiers. This technique firstly proposed fault localization to locate faulty parameters based on uncommon activation functions of neurons. Then it performs repairing using constraint solving by correcting faulty activation functions and solves it as linear programming problem. It focuses on repairing the network logic on the intermediate layer or the last layer. Sun et al have presented CARE [41], a causality-based neural network repair framework, which applies causality-based fault location to all layers of neural networks and performs optimization using PSO algorithm. Also, this framework can repair both FFNNs and CNNs. CARE suggests the causality-based fault localization technique, wherein a neural network is modeled as a structural causal model (SCM), and then the casual contribution to measure the relationship between hidden neurons and predictions is calculated. This technique chooses the faulty neurons based on the casual contribution, which is scalable to various properties of neural networks. Meanwhile, Sun et al [40] present another approach to verify neural networks against fairness. Based on targeted neural networks, this approach learned Markov chains to verify neural networks against fairness via optimization, and it can deal with both FFNNs

and RNNs. BIRDNN [26] is an approach which supports both retraining and fine-tuning. It intends to study the behavior of neurons at bottom, and integrates Monte Carlo sampling technique to local behavior of NNs. Thus the domain repair process is converted to sampled repair process. Similar to CARE, BIRDNN uses PSO algorithm to fix the faulty neurons. Arachne [37] is an innovative search-based repair approach for Deep Neural Networks (DNNs) that addresses specific misclassifications by directly optimizing neural weights through Differential Evolution(DE). It introduces an advanced fault localization technique, Bidirectional Localisation(BL). It demonstrates effectiveness across various datasets and DNN architectures, including handling fairness issues. Despite these strengths, Arachne's method may face challenges with complex model architectures, computational intensity for larger networks, and potential overfitting. RNNRepair [48], a model-based approach for the automatic repair of Recurrent Neural Networks (RNNs). The method's strength lies in its ability to interpret and rectify incorrect behaviors by constructing an influence model that efficiently assesses the impact of training samples on predictions. Nevertheless, it relies on Gaussian Mixture Models for state clustering, which may introduce complexity in determining the optimal number of clusters. Despite this, RNNRepair demonstrates effectiveness in identifying influential training samples and offers a practical solution for enhancing RNN reliability. DeepRepair [50], an innovative approach for the automatic repair of deep neural networks in real-world operational environments. This method leverages style-guided data augmentation to incorporate unknown failure patterns into training data, enhancing the network's robustness against various corruptions. Despite its effectiveness in improving performance across different DNN architectures and failure modes, DeepRepair may demand substantial computational resources and relies on the diversity of collected failure samples. Moreover, its generalizability to unseen failure patterns and adversarial attacks remains an open challenge. VeRe [27] is a verification-guided framework for neural network repair, focusing on identifying and optimizing faulty neurons via linear relaxation. It efficiently enhances model robustness against backdoors and safety violations without performance drawdown. Nevertheless, its application is currently limited to fully-connected layers and formal-specifiable issues. Airepair [38] is a pioneering repair platform for neural networks, enabling the systematic assessment of repair methods. It supports diverse models and datasets, offering a modular framework for direct weight modification, retraining, and architecture extension. Despite its early-stage limitations and potential configuration dependencies, Airepair provides a valuable benchmarking tool for the neural network repair community.

8 LIMITATIONS

Nevertheless, our work has the following limitations:

- **Assumption of Differentiability:** The repair algorithm with constraints relys on the assumption that the DNN's activation functions are differentiable, which may not hold for all types of activation functions used in practice.
- **Scalability:** Although we present a polynomial-time reduction for repair problems, the scalability of the approach to very large and deep neural networks requires further exploration remains an issue. The repair scalability of AutoRIC relies on the solvers of QP currently.
- **Layer-wise Repair:** Limited to the linearization of networks, the implemention of incremental constraint solving with constraints focuses on repairing one layer at a time, which might not capture complex faults that span multiple layers. Although bisectional detection can perform cross-layer repairs, their current performance is mediocre and requires further improvement.
- **Lack of Unified Input Description:** AutoRIC is designed to calculate properties such as fairness, and robustness from case to case. Nevertheless, we need to establish a unified input

description of properties that serves as the input interface for AutoRIC, which remains a challenging issue at present.

In conclusion, while AutoRIC shows significant potential, it is constrained by several limitations. These include the assumption of differentiability, scalability issues with very large and deep neural networks, limitations in layer-wise repair, and the absence of a unified input description framework. Addressing these challenges is crucial for the further development and effectiveness of AutoRIC.

9 CONCLUSION

In this work, we present AutoRIC, an optimization-based technique for repairing neural networks for various properties and the first white-box repair method based on analytical approach. We devise the function between faulty parameters and high-level properties, and convert the repairing problem into an optimization problem. AutoRIC is experimentally evaluated with multiple neural networks trained on benchmark datasets. Results demonstrate that AutoRIC is efficient and effective on repairing the model, accompanied with minor decrease in accuracy of the existing model. Future work could explore the linearization technique on crossing layers of networks. Meanwhile, future work may try to establish a unified input description using specification language of various properties.

10 ACKNOWLEDGMENT

This work is supported by the National Key R&D Program of China No. 2022YFA1005101, the National Natural Science Foundation of China under Grant No. 61872371 and No. 62032024.

REFERENCES

[1] Tameem Adel, Isabel Valera, Zoubin Ghahramani, and Adrian Weller. 2019. One-network adversarial fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 2412–2420.

[2] Aniya Agarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. 2019. Automated Test Generation to Detect Individual Discrimination in AI Models. (2019).

[3] Samuel Benton, Xia Li, Yiling Lou, and Lingming Zhang. 2021. Evaluating and improving unified debugging. *IEEE Transactions on Software Engineering* 48, 11 (2021), 4692–4716.

[4] M. A. Bhatti. 2000. *Practical Optimization Methods With Mathematica Applications*. Springer.

[5] David Boetius, Stefan Leue, and Tobias Sutter. 2023. A robust optimisation perspective on counterexample-guided repair of neural networks. In *International Conference on Machine Learning*. PMLR, 2712–2737.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars.

[7] Cjadams, Jeffrey Sorensen, Julia Elliott, Lucas Dixon, Mark McDonald, nithum, and Will Cukierski. 2017. Toxic Comment Classification Challenge. https://kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge

[8] G. B. Dantzig. 1963. *Linear Programming and Extensions*. Princeton University Press.

[9] Pieter Delobelle, Paul Temple, Gilles Perrouin, Benoît Frénay, Patrick Heymans, and Bettina Berendt. 2021. Ethical adversaries: Towards mitigating unfairness with adversarial machine learning. *ACM SIGKDD Explorations Newsletter* 23, 1 (2021), 32–41.

[10] D. den Hertog. 1993. *Interior Point Approach to Linear, Quadratic and Convex Programming*. Kluwer.

[11] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. 2019. DeepFault: Fault Localization for Deep Neural Networks. In *Fundamental Approaches to Software Engineering - 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11424)*, Reiner Hähnle and Wil M. P. van der Aalst (Eds.). Springer, 171–191. https://doi.org/10.1007/978-3-030-16722-6_10

[12] Feisi Fu and Wenchao Li. 2021. Sound and Complete Neural Network Repair with Minimality and Locality Guarantees. In *International Conference on Learning Representations*.

[13] Xuanqi Gao, Juan Zhai, Shiqing Ma, Chao Shen, Yufei Chen, and Qian Wang. 2022. FairNeuron: improving deep neural network fairness with adversary games on selective neurons. In *Proceedings of the 44th International Conference on Software Engineering*. 921–933.

[14] Sahaj Garg, Vincent Perot, Nicole Limtiaco, Ankur Taly, Ed H Chi, and Alex Beutel. 2019. Counterfactual fairness in text classification through robustness. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. 219–226.

[15] Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. 2020. Minimal Modifications of Deep Neural Networks using Verification.. In *LPAR*, Vol. 2020. 23rd.

[16] Richard Gonzales. 2020. Feds say self-driving uber suv did not recognize jaywalking pedestrian in fatal crash. *NPR https://www. npr. org/2019/11/07/777438412/feds-say-self-driving-uber-suv-did-not-recognize-jaywalking-pedestrian-in-fatal-(Nov 2019), accessed* (2020), 06–06.

[17] Alex Hern. 2017. Facebook translates' good morning'into'attack them', leading to arrest. *the Guardian* 24 (2017).

[18] Kashmir Hill. 2022. Wrongfully accused by an algorithm. In *Ethics of Data and Analytics*. Auerbach Publications, 138–142.

[19] Kyle D Julian, Mykel J Kochenderfer, and Michael P Owen. 2019. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics* 42, 3 (2019), 598–608.

[20] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[21] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS Valentim, Huiying Liang, Sally L Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. 2018. Identifying medical diagnoses and treatable diseases by image-based deep learning. *cell* 172, 5 (2018), 1122–1131.

[22] Ron Kohavi. 1996. Census Income. https://doi.org/10.24432/C5GP7S. https://doi.org/10.24432/C5GP7S Accessed on YYYY-MM-DD.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[24] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[26] Zhen Liang, Taoran Wu, Changyuan Zhao, Wanwei Liu, Bai Xue, Wenjing Yang, and Ji Wang. 2023. Repairing Deep Neural Networks Based on Behavior Imitation. arXiv:2305.03365 [cs.LG]

[27] Jianan Ma, Pengfei Yang, Jingyi Wang, Youcheng Sun, Cheng-Chao Huang, and Zhen Wang. 2024. VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[28] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th international symposium on software reliability engineering (ISSRE)*. IEEE, 100–111.

[29] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 175–186.

[30] Keyvan Majd, Siyu Zhou, Heni Ben Amor, Georgios Fainekos, and Sriram Sankaranarayanan. 2021. Local repair of neural networks using optimization. *arXiv preprint arXiv:2109.14041* (2021).

[31] Sérgio Moro, Paulo Rita, and Paulo Cortez. 2012. Bank Marketing. https://doi.org/10.24432/C5K306. https://doi.org/10.24432/C5K306 Accessed on YYYY-MM-DD.

[32] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[33] Long H Pham, Jiaying Li, and Jun Sun. [n.d.]. SOCRATES: Towards a Unified Platform for Neural Network Analysis. *networks* 16 ([n. d.]), 8.

[34] RT Rockefellar. 1970. Convex analysis Princeton University Press. *Princeton, NJ* (1970).

[35] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. 2018. Fast and Effective Robustness Certification. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 10825–10836. https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html

[36] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitry Pyrkin, Sergei Popov, and Artem Babenko. 2019. Editable Neural Networks. In *International Conference on Learning Representations*.

[37] Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2023. Arachne: Search-Based Repair of Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–26.

[38] Xidan Song, Youcheng Sun, Mustafa A Mustafa, and Lucas C Cordeiro. 2023. Airepair: A repair platform for neural networks. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 98–101.

[39] Matthew Sotoudeh and Aditya V Thakur. 2021. Provable repair of deep neural networks. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 588–603.

[40] Bing Sun, Jun Sun, Ting Dai, and Lijun Zhang. 2021. Probabilistic verification of neural networks against group fairness. In *Formal Methods: 24th International Symposium, FM 2021, Virtual Event, November 20–26, 2021, Proceedings 24*. Springer, 83–102.

[41] Bing Sun, Jun Sun, Long H Pham, and Jie Shi. 2022. Causality-based neural network repair. In *Proceedings of the 44th International Conference on Software Engineering*. 338–349.

[42] Zhe Tao, Stephanie Nawas, Jacqueline Mitchell, and Aditya V Thakur. 2023. Architecture-Preserving Provable Repair of Deep Neural Networks. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 443–467.

[43] Li Tianlin, Cao Yue, Zhang Jian, Zhao Shiqian, Huang Yihao, Liu Aishan, Guo Qing, and Liu Yang. 2023. RUNNER: Responsible UNfair NEuron Repair for Enhancing Deep Neural Network Fairness. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 66–78.

[44] Florian Tramèr, Vaggelis Atlidakis, Roxana Geambasu, Daniel J. Hsu, Jean-Pierre Hubaux, Mathias Humbert, Ari Juels, and Huang Lin. 2017. FairTest: Discovering Unwarranted Associations in Data-Driven Applications. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 401–416. https://doi.org/10.1109/EUROSP.2017.29

[45] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 98–108.

[46] Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, and Corina S Păsăreanu. 2021. Nn repair: Constraint-based repair of neural network classifiers. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*. Springer, 3–25.

[47] Huanhuan Wu, Zheng Li, Zhanqi Cui, and Jianbin Liu. 2022. GenMuNN: A mutation-based approach to repair deep neural network models. *Int. J. Model. Simul. Sci. Comput.* 13, 2 (2022), 2341008:1–2341008:17. https://doi.org/10.1142/S1793962323410088

[48] Xiaofei Xie, Wenbo Guo, Lei Ma, Wei Le, Jian Wang, Lingjun Zhou, Yang Liu, and Xinyu Xing. 2021. RNNrepair: Automatic RNN repair via model-based analysis. In *International Conference on Machine Learning*. PMLR, 11383–11392.

[49] Xiaodong Yang, Tom Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T Johnson, and Danil Prokhorov. 2022. Neural network repair with reachability analysis. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 221–236.

[50] Bing Yu, Hua Qi, Qing Guo, Felix Juefei-Xu, Xiaofei Xie, Lei Ma, and Jianjun Zhao. 2021. Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment. *IEEE Transactions on Reliability* 71, 4 (2021), 1401–1416.

[51] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. 2020. White-box fairness testing through adversarial sampling. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 949–960.