An implementation of fairness guarantee framework for neural networks*

Liu Zihao[1] Hu Shiyu[1]

[1]Peking University, Haidian District, Beijing ,China

Abstract. Neural Network Fairness Framework (NNFF) refers to a set of methods and techniques designed to evaluate and ensure that neural network models are fair to all individuals or groups when making decisions. They work together to reduce or eliminate bias and discrimination in models. In our paper, we apply the built-in neural network trainer and constraint solver in python to give a feasible set of calculations for neural network fairness, with some improvements for different levels of the task for improving it.

Keywords: Fairness Framework · ε-δ-IF · neural networks.

1 Introduction

The earliest work on neural network fairness can be traced back to a 2016 article[1] in which the authors found some gender and racial bias through word embedding models such as Word2Vec. And in another article from October of that year[2], the concept of Equality of Opportunity (EO) and its counterpart were formalized, marking the beginning of research in related fields.

Specifically, the Neural Network Fairness Framework (NNFF) refers to a set of methods and techniques designed to evaluate and ensure that neural network models are fair to all individuals or groups when making decisions. This framework can contain different components, including fairness metrics, training strategies, regularization methods and algorithms, and validation and testing tools, that work together to reduce or eliminate bias and discrimination in models.

A large number of relevant experiments have shown that the following elements are a few key components that make up the fairness assurance framework of neural networks:

Fairness metrics: define quantitative measures of fairness, such as Group Fairness metrics, (specific examples are equal opportunity, equalized parity ratio, etc.); or Individual Fairness metrics (a typical case is ε-δ-IF, which we discuss

* Completion time: **2023/12/13**

in this essay.)

Data processing: prior to model training, the dataset is preprocessed to eliminate or minimize biases in it. This may include resampling, weight adjustment, or using specific dataset transformation techniques.

Fairness-optimized training strategies: modifying the training algorithm to optimize the fairness metric, which may include introducing a fairness regularity term in the loss function or using adversarial training methods to reduce the model's reliance on sensitive attributes.

Model validation and testing: use different evaluation metrics and test sets to check the fairness of the model. This can include cross-validation, adversarial testing, or other robustness tests.

Fairness audit and ongoing monitoring: Conduct a fairness audit after the model is deployed and continuously monitor the model's performance to ensure long-term fairness.

Feedback and improvement: Based on the results of the fairness assessment, models and strategies are adjusted to continuously improve fairness.

Indeed, the purpose of the Neural Network Fairness Guarantee framework is to ensure that the model does not produce unfair results in all relevant aspects (e.g., gender, race, age, etc.), and in this way we can better ensure that the decisions of AI systems are fair and equitable.

In this paper, we adopt the "ε-δ-IF" definition of individual fairness from Elias, Andrea et al. in "Individual Fairness Guarantees for Neural Networks"[3], and apply the python machine learning library and the PULP open-source solver to solve the fairness test of the Boston house price dataset containing the independent variable of possible discrimination bias of "ethnic color" to a certain extent.

At the same time we will further improve the fairness performance in two different dimensions, the provided dataset and the training dataset, by implementing a resampling of the dataset and a midway adversarial training method.

2 Related Work

Deep learning algorithms have made great strides and are increasingly being used in decision-making applications that affect the lives of individuals, including image classification, speech understanding, self-driving medical diagnostics, and more. However, the application of deep learning to sensitive attributes such as race, age, gender, etc. based on attributes remain inequitable, and this data-based learning learning methods can overly correlate sensitive attributes and

may show discriminatory behavior towards protected groups discriminatory behavior towards protected groups.A clear example in society is the existence of discriminatory behavior towards the sensitive attribute of gender in certain CV screening tools, resulting in men having an advantage over women in the job application process.

2.1 Bias in training

Bias arises from two sources, data and model bias. For training data, in which there may be biases arising from historical social causes, models learned on biased data may lead to unfairness in prediction results. Suresh et al.[4] discuss the different sources of data bias and how these biases arise. Oheanu et al.[5] prepared a relatively complete list of different types of bias and analyzed the consequences due to data bias.

As for model bias, it is triggered by subtle differences in the way the deep learning algorithms themselves work, and these differences may cause the deep models to make unfair decisions. And one of the most widespread types of bias is the bias that discriminates against predicted outcomes. Discrimination refers to the fact that due to the membership of certain groups, deep models model produces unfavorable decision outcomes for members of these groups due to membership in certain groups. Deep learning is based on a data-driven learning paradigm that enables models to automatically automatically learn useful representations from data that may contain biases, which can lead to deep models replicating the data. bias, which can cause deep models to replicate and even amplify the bias that exists in the data. This can lead deep models to replicate and even amplify the biases present in the data, making unfounded associations that tend to amplify stereotypes about certain sensitive attributes.More details of the categorization can be found in an article from Du et al[6].

2.2 De-biasing methods

A number of de-biasing methods have been devised to address the above bias problem.Common data-level de-biasing methods include the data resampling method of Burnaev et al[7]; the category-equalized sampling method of Cui et al[8], the SMOTE synthetic data method of Chawla et al[9], and the data enhancement method[10]. As for the algorithmic level of debiasing, cost-sensitive debiasing methods based on cost-sensitivity[11] is a common way to deal with it, and some of the most recent articles will also try to use principal component analysis.

Deep learning model debiasing then determines the fairness of training the model itself after data preprocessing. Debiasing methods based on model regularization are the most popular paradigm in classical machine learning. Common regular

terms are inscribed as follows:

$$L(\theta, x, y, r) = d_1(y, \hat{y}) + \lambda_1 d_2\left(f_{loc}(x), r\right) + \lambda_2 R(\theta)$$

where d_1 is the normal classification loss function and $R(\theta)$ is the regularization term, the function $f_{loc}(x)$ is the local interpretation method, and d_2 is the distance metric function. Each of these three terms is used to guide the deep model to make correct predictions, and the hyperparameters λ_1 and λ_2 are used to balance these three terms.

A more practical approach to optimization is the de-biasing method based on adversarial training.From the perspective of model training, adversarial training is a typical solution to remove information about sensitive attributes from the intermediate representation of a deep model in order to obtain a fair classifier . The goal is to learn a high-level input representation that has the maximum amount of information for the main pre measurement task with the maximum amount of information and the protected attributes with the least predictive.

2.3 Metrics of fairness

Metrics of fairness have been something that needs to be addressed urgently: there are a number of well-known debiased experimentation platforms on the market: such as Microsoft's Fairlearn, IBM's AI Fairness360, and Google's ML-fairness-gym, but there is still no consensus on fairness metrics. In some cases, some metrics may conflict with others, and a model may be fair on one metric but often lead to other types of unfairness. Currently good metrics such as average prediction, sample distortion, and generalized entropy index can serve as a guide for fairness optimization to some extent, but whether they are strictly guided and that metric can be applied more broadly are questions that are still unclear.

3 Important concepts

Our entire experiment is based on the definition of Individual Fairness and its constrained characterization of the problem in the paper by Elias et al[3].Some important concepts will be explained clearly in this section.

3.1 Neural Networks for Applications

In the reference essay, the writers consider an L layer fully-connected NN $f^w : X \to Y$, parameterised by a vector of weights $w \in R^{n_w}$ trained on $\mathcal{D} = \{(x_i, y_i), i \in \{1, \ldots, n_d\}\}$. For an input $x \in X, i = 1, \ldots, L$ and $j = 1, \ldots, n_i$, the NN is defined as:

$$\phi_j^{(i)} = \sum_{k=1}^{n_{i-1}} W_{jk}^{(i)} \zeta_k^{(i-1)} + b_j^{(i)}, \quad \zeta_j^{(i)} = \sigma^{(i)}\left(\phi_j^{(i)}\right) \tag{1}$$

where $\zeta_j^{(0)} = x_j$. Here, n_i is the number of units in the i th layer, $W_{jk}^{(i)}$ and $b_j^{(i)}$ are its weights and biases, $\sigma^{(i)}$ is the activation function, $\phi^{(i)}$ is the pre-activation and $\zeta^{(i)}$ the activation. The NN output is the result of these computations, $f^w(x) := \zeta^{(L)}$. In regression, $f^w(x)$ is the prediction, while for classification it represents the class probability.

3.2 Individual Fairness

Individual Fairness (IF) means that for each pair of individuals in the input space, the model should guarantee a fair treatment, i.e., if two individuals are similar, they should get similar predictions. This notion of fairness is based on a worst-case metric, and its guarantee applies to every possible individual in the input space. The opposite is **group fairness**, which focuses on the statistical properties of the model with respect to the demographic characteristics of the data. It is noted in the referenced article that while group fairness is more widely adopted, it can lead to a situation where the model, while fair to the group as a whole, may be perceived as unfair to a particular individual.

3.3 ε-δ-IF

In an article of Verifying individual fairness(John et al., 2020)[12],it mentions a method called The ε-δ-IF formulation:

Definition 1(ε-δ-IF formulation) Consider $\varepsilon \geq 0$ and $\delta \geq 0$. We say that f^w is ε - δ-individually fair w.r.t. d_{fair} iff

$$\forall x', x'' \text{ s.t. } d_{\text{fair}}\ (x', x'') \leq \varepsilon \Longrightarrow |f^w\,(x') - f^w\,(x'')| \leq \delta \tag{2}$$

Here, ε measures similarity between individuals and δ is the difference in outcomes (class probability for classification). We emphasise that individual fairness is a global notion, as the condition in Definition 1 must hold for all pairs of points in input space.

The similarity metric d_{fair} is a key component of Eq. (2) used to quantify the similarity between input individuals in the individual fairness certification of neural networks. The choice of dfair is varied, subject to the condition that it can be efficiently encoded by mixed integer linear programming (MILP) problems. In particular, the following metrics are considered:

Weighted ℓ_p (We apply it to our experiments). In this case $d_{\text{fair}}\ (x', x'')$ is defined as a weighted version of an ℓ_p metric, i.e. $d_{\text{fair}}\ (x', x'') = \sqrt[p]{\sum_{i=1}^{n} \theta_i \, |x_i' - x_i''|^p}$. We address the issue of sensitive attributes—such as race or gender—by assigning them a weight of zero in our calculations. This approach ensures that two individuals are deemed equivalent when their differences are solely in these sensitive dimensions. For the other attributes that are not sensitive, we assign weights,

denoted by θ_i, which can be adjusted to reflect the extent to which each feature is associated with the sensitive attributes. By doing so, we can fine-tune the importance of each non-sensitive feature in the overall measure of similarity, allowing for a more nuanced and fair comparison between individuals. This strategy is particularly useful in creating metrics that aim to support individual fairness by preventing sensitive characteristics from influencing the assessment of similarity.

Mahalanobis. In this case we have $d_{\text{fair}}\left(x', x''\right) = \sqrt{\left(x' - x''\right)^T S \left(x' - x''\right)}$, for a given positive semi-definite (SPD) matrix S. The Mahalanobis distance generalises the ℓ_2 metric by taking into account the intra-correlation of features to capture latent dependencies w.r.t. the sensitive features.

Feature Embedding. The metric is computed on a proper embedding, so that $d_{\text{fair}}\left(x', x''\right) = \hat{d}\left(\varphi\left(x'\right), \varphi\left(x''\right)\right)$, where $\hat{d}$ is either the Mahalanobis or the weighted ℓ_p metric, and φ is a feature embedding map. These allow for greater modelling flexibility, at the cost of reduced interpretability.

3.4 Problem Formulation

Problem 1 (Fairness Certification). Given a trained NN f^w, a similarity d_{fair} and a distance threshold $\varepsilon \geq 0$, compute

$$\delta_{\max} = \max_{\substack{x', x'' \in X \\ d_{\text{fair}}\left(x', x''\right) \leq \epsilon}} \left| f^w\left(x'\right) - f^w\left(x''\right) \right|. \tag{3}$$

Defining Problem 1 as an optimization problem allows us to use existing optimization techniques, such as Mixed Integer Linear Programming (MILP), to find a solution to the problem and thus achieve fairness guarantees for neural networks. Also the definition of Problem 1 is not limited to specific types of neural networks or specific application scenarios, it provides a generalized approach to evaluate and guarantee individual fairness for many types of neural network models.

Problem 2 (Fairness Training). Consider an NN f^w, a training set $\mathcal{D}$, a similarity metric d_{fair} and a distance threshold $\epsilon \geq 0$. Let $\lambda \in [0, 1]$ be a constant. Define the IF-fair loss as

$$L_{\text{fair}}\left(f^w\left(x_i\right), y_i, f^w\left(x_i^*\right), \lambda\right) = \lambda L\left(f^w\left(x_i\right), y_i\right) + (1 - \lambda)\left|f^w\left(x_i\right) - f^w\left(x_i^*\right)\right| \tag{4}$$

where $x_i^* = \arg\max_{x \in X} \text{ s.t. } d_{\text{fair}}\left(x_i, x\right) \leq \epsilon \left|f^w\left(x_i\right) - f^w(x)\right|$. The ϵ-IF training problem is defined as finding w^{fair} s.t.:

$$w^{\text{fair}} = \arg\min_w \sum_{i=1}^{n_d} L_{\text{fair}}\left(f^w\left(x_i\right), y_i\right). \tag{5}$$

The goal of Problem 2 is to find the weight w_{fair} which minimizes the IF-fair loss at all training points. By adjusting λ, we can weigh the accuracy and fairness of the model during the training process, and thus train a neural network model that performs well in real-world applications and is also fair.

Problem 2' (Fairness Training W/O accuracy). Letting the value of λ in problem 2 be 1 translates into the main concern of our paper: focusing on regulating the fairness of the model without letting it be controlled by accuracy.This is simple to handle and at the same time better allows us to validate the methods we implement.

3.5 Piecewise-linear (PWL) techniques

Piecewise-linear (PWL) techniques provide a powerful tool for approximating non-linear functions, which are ubiquitous in various fields, including machine learning, optimization, and control theory. A PWL function is composed of multiple linear segments, each defined over a specific interval of the input domain. These segments are connected end-to-end, forming a continuous function that approximates a given non-linear function.

PWL approximations are particularly useful when dealing with non-linear activation functions in neural networks. By over-approximating non-linear functions with PWL bounds, one can transform an originally non-convex optimization problem into a Mixed-Integer Linear Programming (MILP) problem. This enables the use of efficient solvers that can handle linear constraints to approximate the behavior of complex non-linear models.

In the context of neural networks, a typical PWL approximation would involve constructing upper and lower bounding lines for each segment of the non-linear activation function. This is achieved by selecting grid points over the range of the activation function and computing the corresponding bounding lines, either through the tangent at the midpoint (for a convex or concave region) or by connecting the endpoints of the interval. The PWL bounds are then encoded into the MILP framework, allowing for the approximation of the neural network's behavior within certain error bounds.

4 Implementation of the framework

4.1 Applied Models and Data

We apply the Boston housing price dataset to discuss this issue primarily: the Boston Housing dataset (BHDS) is a well-known dataset commonly used for teaching and research in supervised learning and regression analysis. It contains information on housing in the Boston area of the United States and is mainly used to predict the median house price. The characteristics are shown in the

table below, where the first 13 characteristics are the independent variables that may affect the final prediction, and "MEDV" is the dependent variable for the final prediction.

CRIM	Per capita crime rate by town.
ZN	Proportion of residential land zoned for lots over 25,000 square feet.
INDUS	Proportion of non-retail business acres per town.
CHAS	Charles River dummy variable (1 if tract bounds river; 0 otherwise).
NOX	Nitric oxides concentration (parts per ten million).
RM	Average number of rooms per dwelling.
AGE	Proportion of owner-occupied units built prior to 1940.
DIS	Weighted distances to five Boston employment centres.
RAD	Index of accessibility to radial highways.
TAX	Full-value property-tax rate per $10,000.
PTRATIO	Pupil-teacher ratio by town.
B	$1000(B_k - 0.63)^2$, where B_k is the proportion of blacks by town.
LSTAT	Percentage of lower status of the population.
MEDV	Median value of owner-occupied homes in $1000s.

Table 1. Description of the variables in the Boston Housing Dataset.

The characteristic "B" reflects the proportion of blacks in the town as a percentage of all (with an added step of quadratic treatment), an irrelevant variable that we intend to exclude.As for the independent variable, there may be a big deviation in the range of the component intervals, so we standardized the component data in the pre-processing of the data, and solved the problem of unbalanced weights by using the built-in scaler in python, and the processed data were attached to the file "Boston_X_scaled.csv"

We use a weighted L2-paradigm in designing the sample similarity measure of the independent variables, in which the weight of the "B" feature term is constant at 0. For the other weights, we apply OpenAI's large language model GPT4 to identify the relevance of the weights to the problem for us, and assign weights to these weights in an isotropic series pattern (in the real-world problem, the weights have to be chosen carefully, and we are only simulating the process of a measure here).Its application in python is similar to a dictionary pattern:

```
A=[{'RM': 0.14444444444444446, 'LSTAT': 0.13333333333333336, 'INDUS': 0.12222222222222222,
    'NOX': 0.11111111111111109, 'CRIM': 0.09999999999999999, 'AGE': 0.08888888888888889,
    'DIS': 0.07777777777777778, 'TAX': 0.06666666666666668, 'PIRATIO': 0.055555555555555546,
    'ZN': 0.044444444444444446, 'RAD': 0.033333333333333326, 'CHAS': 0.022222222222222223, 'B': 0.0},
]
```

Fig. 1. A python dictionary to simulate the weights to house price forecasts.

For simplicity, we trained a single-layer fully-connected neural network (where the activation function is the affine post-sigmoid function) to verify the validity of the fairness measure.The model can be simplified as

$$g(x) = 5 + 45 \times \text{Sigmoid}(wx + b) \tag{6}$$

4.2 Verify the realization of the fairness measure

As mentioned in the article [3], the MILP solver may be very slow to solve nonlinear functions for very large samples, and although our dataset is a very small amount, we still simulate a linear PWL carve-out for nonlinear functions such as the Sigmoid function.

As for the linear solver, we chose python's built-in PuLP solver model and applied an upper and lower linear Sigmoid fit to inscribe the constraints as the maximum difference between the upper and lower linear Sigmoid values at two points.(i.e. delta_val = max(abs(y_pred_upper[i,j] - y_pred_lower[j,i]))).

4.3 Selection of a more equitable model

Since model training with accuracy in mind is too complex and not something we need to do for this task, we only make some adjustments to our parameter choices in terms of "improving fairness". It is worth noting that improving the fairness will reduce the generalization ability of the model to some extent.

We seek the optimal fair model that we hope to be able to get step by step by using pre-processing of data before training and simulated gradient descent during training, while illustrating that some of the methods are strongly optimized at the overall level, and some of them are only helpful at the fine-tuning level. At the same time we are able to give a good algorithmic framework for finding fair models based on these methods.

5 Experiment and Result

5.1 Certificate Steps

In order to facilitate the observation of the data, after many experiments of trial and error, we finally chose an experimental sample with ε in the range of 0.1 to 0.5, and all of our experimental data in the next major section are designed based on this criterion.

First we have to determine a suitable ε-bound for the ε-δ-IF division, which requires us to randomize the parameters and iteratively verify them in $\varepsilon \in [0.1, 0.5)$.Here are the results after 3 random samples:

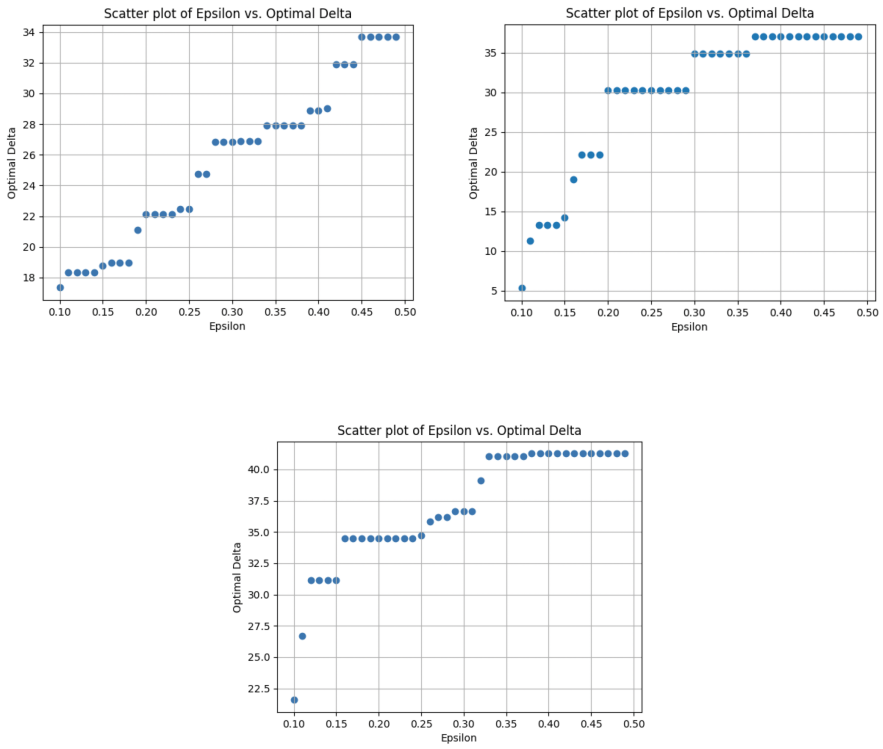

Fig. 2. ε-δ-IF measure under stochastic parameters

We find that the ε-δ-IF values obtained after the initial values of the parameters are chosen differently by randomization are very different. This is well explained because the ε-δ-IF measure is not standard linear. For relatively standard models, the constrained solved δ almost satisfies a linear distribution as ε increases (first figure), while in some special cases (e.g., the glitches in the second and third figure) the constraint solved ε-δ-IF value is very different in this type of model, the points of the next sample distribution may be very close to each other, leading to a distribution similar to a logarithmic function.

For this reason we identified two different directions that can be optimized: one direction of optimization is that we find that model whose distribution tends to be most linear over the feasible intervals of a given demand. Such a distribution ensures that there is a relatively stable change in the value of δ within each ε-varying interval, and thus is a more explanatory model. Another direction of optimization is to find the kind of model where ε is smaller with ε fixed as much as possible, which suggests that we can use a much larger ε to meaningfully de-

scribe the degree of variation in the samples, which numerically is a much better quality model.

In fact, optimization in the first direction is difficult: first of all, computing the change in the δ segment corresponding to a ε segment by constraint solving is inherently intractable; then we don't have an effective tool for fitting the change in the number of functions of the data points in the change; and finally, if we do get a linear fit, but it is satisfied only for a very small δ segment, such a model will not be of wide applicability.

So we chose the second optimization direction to address our problem, and at the same time, we chose the median ε=0.3 as the benchmark to optimize the model by optimizing its corresponding δ value. We conducted 20 random parameter generation on this benchmark, and the experimental results of the δ values obtained are as figure 3.

We have processed the experimental data in descending order, and we can see that the range of δ values after random sampling is selected is very wide, with the smallest value being close to 10 and the largest value being more than 40.Thus, we realize that the results will inevitably be very poor if we preemptively start our optimization at a very poor point, and thus a simple step to simplify it is to start the optimization task by Perform an appropriate number of random samples and select the parameters w and b corresponding to the appropriate/minimum δ values as the startup items for the optimization.

We sample a set of parameters w and b from which the exact values are shown in Figure 4. Corresponding to $\delta = 17.38612$ when $\varepsilon = 0.3$.

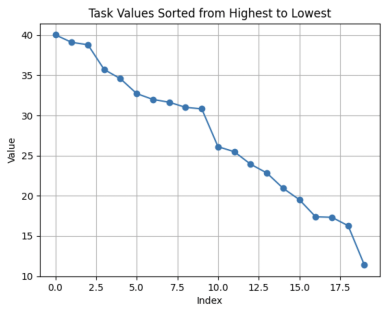

Fig. 3. Changes in δ value at ε =0.3 (20 random samples)

```
>>> w
array([ 1.04279199,  1.46830472,  1.09590334,  1.09480639,  2.28170549,
       -1.06148196,  0.10097893,  0.48551476,  2.56282473,  0.5596913 ,
       -1.25358298,  0.37576072,  1.87801127])
>>> b
-0.55837739849488
```

Fig. 4. Selected parameters w and b

5.2 Resampling Steps

In order to get fairer NNs, we can apply resampling methods to change the distribution of the training data[7]. First, we standardlize the sensitive attribute and the output variable. Second, we compute the pseudo weights to emphasize the outliers of the sensitive attribute and compute the pseudo mean output assuming that the sensitive attribute of the training data is independent from other attributes. Thirdly, we compute the output weights to emphasize the inliers of the output variable refer to the pseudo mean output value and mutiple these two weights to get the final resampling weights. Finally, we use the resampling weights to perform random sampling with replacement and get the resampled training data with the same size of the original training data.

5.3 Simulated gradient optimization Steps

For a problem that needs to be solved by constraint solving techniques, it would be difficult to treat it as an optimization term similar to a "penalty function" and then apply a layer of optimization on it (equivalent to solving a minimax problem). For this reason, we apply the simplicity of linearly fitting the gradient of a sigmoid function, and at the same time, we use the relationship between difference and differentiation to discretely simulate the gradient of a varying parameter term, which, in the next experiments, will be seen to have a certain optimization effect on the variables that have a very poor initial situation but may have a counter-acting effect on those that have a good initial situation.

Since the lower_sigmoid function directly reflects the fractional gradient values of the original activation function itself, we have appropriately simplified our definition of the difference, and instead of using the upper_sigmoid function, we have gone directly to the definition of delta_val = abs(y_pred_lower[i] - y_pred_lower[j]).

In fact, in the code we implement the steps "estimate_gradient" and "gradient_descent", which is not very different from the gradient descent mentioned in general numerical analysis books. It is worth mentioning that we can only take a small learning rate (0.01 in the following experiments) to solve this problem, which may be related to the high singularity of the loss function corresponding

to this constraint satisfaction problem.

The experimental data is shown in the following figure, we randomly selected a starting data point(Due to the perturbation property of the magnitude of the change, we only consider the case where the initial optimization has been optimized to a good extent.) and used 10 rounds of simulated gradient iterations on them, through the experimental figure, we found that the simulated gradient does play a certain role of gradient descent, however, the magnitude of the gradient descent is not obvious, and at the same time by the influence of the step size, not every step is descending, and even the learning rate of 0.01 can not guarantee that this, thus This simple method is only suitable for inspiring fine-tuning in short process steps.

From the experiment in Figure. 5.1, we can see that when the initial parameters w and b are well chosen so that the initial δ is around 6.6 without resampling, we can still make the oscillations drop to smaller values in relatively short steps by appropriate gradient descent methods (the data in the figure reaches a better value of 6.18 at Epoch 7). However it is not guaranteed that each iteration is descending at each step, and at the same time when the learning rate is not met there can be quite large error offsets such as Epoch 10 (in fact the figure is shifted directly to 44.75, which means that at this point w and b can barely differentiate between the data at $\varepsilon \geq 0.3$, which is clearly a bad offset.

With resampling, we may get better outcomes, in Figure. 5.2, we can see that the initial δ is around 6.0 and the best δ appears at Epoch 5.

We can make a predictable conjecture based on these phenomena: as shown in Figure 6, our envisioned modeling of the constraint solving model inscribed as a function about $w - b - \delta$ may be quite exotic. One possible structure is the toothbrush-like structure in the figure, where an initial start to the trough may converge to a relatively small local value in very few sub-steps, whereas too many restarts will directly allow δ to reach the worst possible result of an almost maximum value due to the singularity of the function itself. In this case further fine-tuning with gradient descent just doesn't make sense.

5.4 An Interpretable Framework for Fairness Optimization

Combining the above approaches, we can integrate a generalized practice scheme for improving the fairness of models.It is written in the form of the following algorithms(main,Resampling,Simulated gradient).

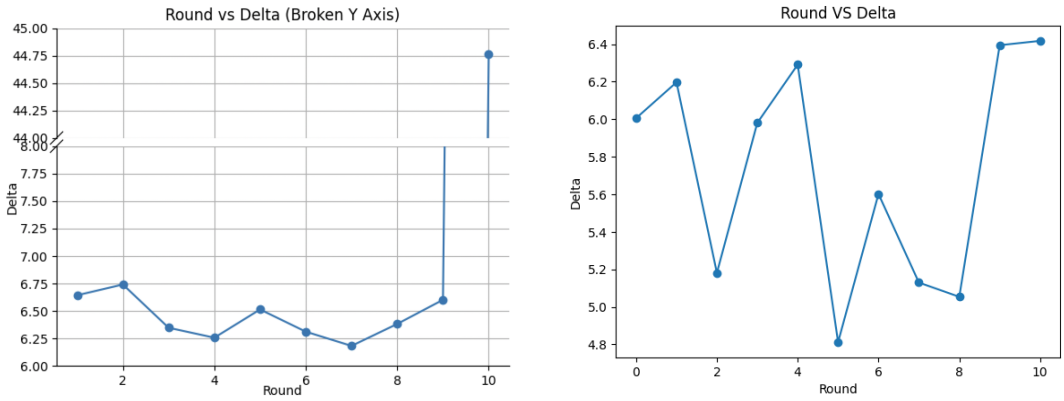

Fig. 5. Optimal Delta via Epochs

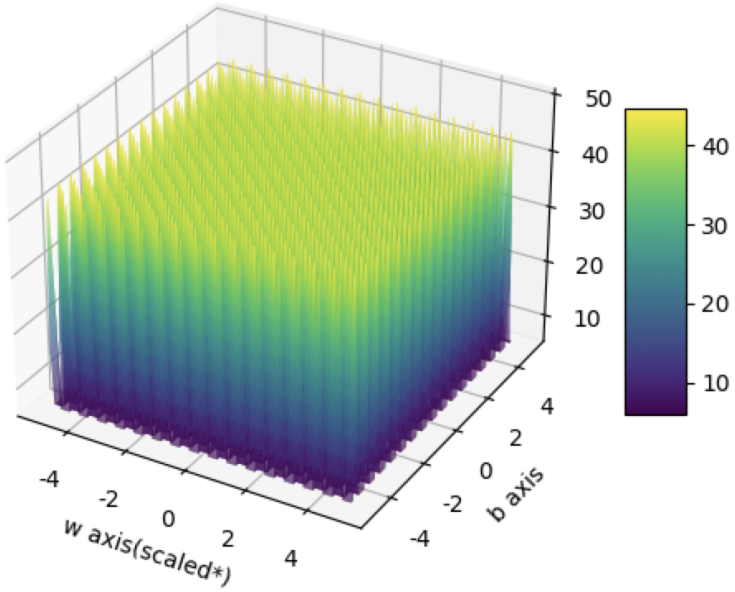

Fig. 6. Conjectural modeling of the $w - b - \delta$ function

Algorithm 1: Interpretable Framework for Fairness Optimization

Input: $X\_$ scaled,NN Model,ε
Output: $X\_$ tuned,$w_{\text{selected}}, b_{\text{selected}}, \delta_{\text{selected}}$

initialization
$w_i, b_i, \delta_i \leftarrow RandomSearch(\text{PuLP}(X\_ \text{ scaled})), i = 20 \sim 40$
$w^*, b^*, \delta^* = \text{argmax}_{w_i, b_i, \delta_i} \delta_i$
$X\_$ tuned $\leftarrow Resampling(X\_ \text{ scaled}, \text{debiased weighting model})$
$\delta_0 \leftarrow \text{PuLP}(X\_ \text{ tuned}, w^*, b^*)$
$w_{\text{selected}}, b_{\text{selected}}, \delta_{\text{selected}} \leftarrow$ $Simulatedgradient(w^*, b^*, X\_ \text{ tuned}, \delta_0)$,epoch=$10 \sim 20$

Algorithm 2: Resampling

Input: Sensitive attribute f_s, Output variable f_o
Output: Resampling weights w

$\widetilde{f_s} = \frac{f_s - \mu_s}{\sigma_s}, \widetilde{f_o} = \frac{f_o - \mu_o}{\sigma_o}$
$w' = \left|\widetilde{f_s}\right| + 1$
$f' = \frac{sum(\widetilde{f_o} w')}{sum(w')}$
$w = \frac{w'}{\left|\widetilde{f_o} - f'\right| + 1}$

Algorithm 3: Simulated gradient

Input: $w^*, b^*, X\_$ tuned, $\delta_0, Epoch, \varepsilon, h = 10^{-4}$,learning_ rate, $i = 0$
Output: $w_{\text{selected}}, b_{\text{selected}}, \delta_{\text{selected}}$

$w \leftarrow w^*, b \leftarrow b^*, \delta \leftarrow \delta_0$
while $i \leq Epoch$ **do**
  grad_w,grad_b=estimate_gradient(w, b)
  $w -=$ learning_rate * grad_w
  $b -=$ learning_rate * grad_b
  $\delta_{\text{test}} \leftarrow \text{PuLP}(X\_ \text{ tuned}, w, b)$
  if $\delta_{test} >> \delta$ **then**
    $w_{\text{selected}}, b_{\text{selected}} \leftarrow w(\text{pref}), b(\text{pref})$
    break
  else
    $\delta = \max(\delta, \delta_{\text{test}})$, $w_{\text{selected}}, b_{\text{selected}} \leftarrow w(\text{pref}), b(\text{pref})$
    $i++$
  end
end
$\delta_{\text{selected}} = \delta$

Roughly, the steps of the Algorithm 1 are as follows: first, randomly search the whole space for about 20-40 rounds, and select the parameter w, b that corresponds to the smallest value of δ; then, use the resampling technique (Algorithm 2) to exclude the data that we think are biased in the sampling, and obtain the fine-tuned set of independent variables; finally, apply the algorithm of simulat-

ing the gradient(Algorithm 3) , and make the decibel-level adjustment of the value of δ, and if there is any case of the large learning rate jumping to the high point, then we will directly truncate the loop, and select the parameter w, b that corresponds to the smallest value of δ in the previous epochs.

6 Summary and Future thoughts

In this paper, we adopt the ε-δ-IF measure representation of the neural network fairness assurance framework from the paper [3] and implement it on a regression set of Boston house price dataset that possesses some error bias. We develop a validation-optimization model through which we select the ε appropriate for studying fairness efforts, and provide an algorithm to optimize as much as possible the relatively smaller δ corresponding to a single point of ε, achieving an average of about 60% optimization with respect to sampling randomly at one time.

We acknowledge that this implementation is simple and that there are still many directions in which optimization could be investigated. For example, when randomly selecting the initial data, can we add a predictable functional distribution based on completely randomly selecting out the initial optimization term, which can allow us to start our optimization step at multiple relatively small values of δ; for example, is there any better sampling restriction that can exclude bias factors in the re-sampling process; and for example, in the process of proposing the gradient descent, in addition to the application of the reduced learning rate, how to get rid of the situation where the descending gradient jumps directly to the high point without archiving (e.g., random restart or annealing, or try to pick the appropriate direction to take the maximum of the descent in the small critical domain near the direction of the gradient, etc.). The implementation of these features would require much larger/more biased datasets and more complex code to assist in doing so.

Alternatively, we are simplifying our consideration of this issue at a level that does not take into account the predictive accuracy of the model at all. However, the trade-off between fairness and accuracy of neural network models is a widely discussed issue in the field of machine learning. In many cases, improving the fairness of a model may lead to a loss of accuracy. In order to ensure fairness, measures are taken to reduce the model's reliance on sensitive features, which, while enhancing fairness, may affect the model's ability to capture other important patterns in the data because they limit the model's ability to learn using the full range of data. For example, if resampling causes certain important but unbalanced features to be overlooked, the model may not be able to fully understand the relationship between those features and the predicted outcomes, thus affecting accuracy. In addition, if the loss function is modified to penalize the reliance on sensitive features, the model may make decisions without considering these features, which may ignore patterns that are indeed relevant to the

prediction goal but are associated with sensitive features.

During model training, a multi-objective optimization approach can be taken to consider both accuracy and fairness. For example, accuracy and fairness can be considered as two objectives and their importance can be weighed in the loss function. The most common approach is to use "accuracy-fairness" co-tuning based on controlling small neighborhoods, i.e., based on our proposed fairness guarantee framework, we add an accuracy tuning step, and then decide the order and number of repetitions of the steps in accordance with the ratio (i.e., the allocation of λ in Problem 2).

Overall, our implementation provides the right research direction to solve this huge problem. More complex algorithms require larger datasets and more time to try and adapt.

7 Division of tasks and acknowledgements

Liu Zihao and Hu Shiyu did different tasks to accomplish this topic: Liu Zihao was mainly responsible for the selection of experimental methods and measures, the implementation of the validation part and the code to simulate the gradient descent, as well as the writing of the whole paper. Hu Shiyu was mainly responsible for researching related reference papers, designing the resampling method, and organizing the experimental data.

We would like to express our gratitude to Mr. Sun Meng, who provided us with the overall idea, to Senior Zhang Yihao, who helped us to organize our thoughts, and to every one of the students who read our articles before uploading the assignment.

Our code will be attached to the GitHub :
https://github.com/Zihao-Liu-Leo/FairNN-Co-Shiyu-Hu

References

1. Caliskan, Aylin, Joanna J. Bryson, and Arvind Narayanan: Semantics derived automatically from language corpora contain human-like biases. Science 356.6334, 183–186 (2017)
2. Moritz Hardt, Eric Price, and Nathan Srebro: Equality of Opportunity in Supervised Learning. Advances in Neural Information Processing Systems 29, (2016)
3. Elias Benussi, Andrea Patane, Matthew Wicker, Luca Laurenti, and Marta Kwiatkowska: Individual Fairness Guarantees for Neural Networks. International Joint Conference on Artificial Intelligence Main Track 31, 651–658 (2022)
4. Suresh Harini, and John Guttag: A framework for understanding sources of harm throughout the machine learning life cycle. Equity and access in algorithms, mechanisms, and optimization, 1–9 (2021)

5. Olteanu Alexandra, Emre Kıcıman, and Carlos Castillo: A critical review of online social data: Biases, methodological pitfalls, and ethical boundaries. Proceedings of the eleventh ACM international conference on web search and data mining, (2018)
6. Mengnan Du, Fan Yang, Na Zou, and Xia Hu: Fairness in deep learning: A computational perspective. IEEE Intelligent Systems 36.4, 25–34 (2020)
7. Burnaev Evgeny, Pavel Erofeev, and Artem Papanov: Influence of resampling on accuracy of imbalanced classification. Eighth international conference on machine vision Vol. 9875. SPIE, (2015)
8. Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie:Class-balanced loss based on effective number of samples. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, (2019)
9. NV Chawla, KW Bowyer, LO Hall, and WP Kegelmeyer: SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research 16, 321–357(2002)
10. Shorten Connor, and Taghi M. Khoshgoftaar: A survey on image data augmentation for deep learning. Journal of big data 6.1, 1–48(2019)
11. John Philips George, Deepak Vijaykeerthy, and Diptikalyan Saha: Verifying individual fairness in machine learning models. Conference on Uncertainty in Artificial Intelligence. PMLR, (2020)