# Fast & Fair: Efficient Second-Order Robust Optimization for Fairness in Machine Learning[*]

Allen Joseph Minch[†], Hung Anh Dinh Vu[‡], and Anne Marie Warren[§]

*Project Advisor: Dr. Elizabeth Newman*[¶]

**Abstract.** This project explores adversarial training techniques to develop fairer Deep Neural Networks (DNNs) to mitigate the inherent bias they are known to exhibit. DNNs are susceptible to inheriting bias with respect to sensitive attributes such as race and gender, which can lead to life-altering outcomes (e.g., demographic bias in facial recognition software used to arrest a suspect). We propose a robust optimization problem, which we demonstrate can improve fairness in several datasets, both synthetic and real-world, using an affine linear model. Leveraging second order information, we are able to find a solution to our optimization problem more efficiently than with a purely first order method.

**Key words.** machine learning, fairness, robust optimization, adversarial training, optimization

**MSC codes.** 65F10, 65F22, 65K05, 90C47

## 1. Introduction.

Machine learning has become an integral part of data analysis with its powerful ability to reveal underlying patterns and structures in data. Deep Neural Networks (DNNs) in particular are the gold standard classifying complex data; however, there is a tendency for DNNs to inherit bias from the datasets on which they train. Bias in this sense is not statistical bias, but the ways in which individual advantages or disadvantages manifest in data. This can be especially problematic in areas where machine learning is used to make life-altering decisions such as criminal justice [4] and corporate hiring [7].

The ever-expanding use of machine learning poses a significant ethical question when models are known to perpetuate societal biases [7]. While an in-depth discussion of these ethical concerns is beyond the scope of this work, they motivate our efforts to improve fairness within the models themselves. The unfortunate truth is that the harmful biases we see in our models and in our data come from deep-rooted societal structures that are at present beyond the abilities of machine learning to correct. However, we feel that in the face of these larger issues it is our duty within our means to work towards fairer outcomes.

One way to potentially achieve fairer outcomes is to use adversarial training to introduce robustness to a model. Robust optimization aims to make the model less susceptible to small variations in data, known as adversarial attacks, but in doing so decreases model accuracy. Recently, the Fair-Robust-Learning framework was proposed to reduce this unfairness problem in adversarial training [13]. The authors demonstrated that a combination of fairness and adversarial regularization yielded fairer models on benchmark image classification datasets.

[*]Submitted to the editors January 31, 2024.

**Funding:** This work is supported in part by the US NSF award DMS-2051019.

[†]Department of Mathematics, Brandeis University, Waltham, MA (allenminch@brandeis.edu)

[‡]Department of Mathematics, University of Maryland, College Park, MD (hvu1@terpmail.umd.edu)

[§]University of Minnesota, Minneapolis, MN (warre659@umn.edu, https://anniewarren.github.io)

[¶]Department of Mathematics, Emory University, Atlanta, GA (elizabeth.newman@emory.edu).

Our research shares the goal of addressing fairness issues in DNNs through the use of adversarial training techniques, but focuses on an additive bias rather than out-of-distribution bias or other forms. We define fairness on different metrics (independence, separation, and sufficiency vs. average and worst-class boundary, robust, and standard errors) to measure additive bias with respect to sensitive 'hidden' attributes. Without aiming to cater to specific types of data, we explore the effects of adversarial training on this definition of fairness. A simultaneous focus is to improve the efficiency of solving robust optimization problems. To this end, we use second-order information to accelerate training, a concept that was not addressed in previous work [13].

We implement a second-order method, termed the "trust region subproblem" (TRS), designed explicitly to address inner optimization challenges encountered when introducing robust training. Our experiments, spanning both synthetic and real-world datasets, demonstrate the capabilities of robust optimization in enhancing fairness. We employ three distinct optimizers for these tests, allowing us to compare their performance. Notably, the integration of `hessQuik` [8], has proven instrumental in efficiently deriving exact Hessians. This approach surpasses the projected gradient descent (PGD) method in terms of time efficiency while producing the same solution. For transparency and further community engagement, we've made our Python implementation, including all our experiments, available on our GitHub repository at Fast-N-Fair (https://github.com/elizabethnewman/fast-n-fair).

The paper is organized as follows: Section 2 introduces DNNs and the necessary notation, robust optimization, and our choice of fairness metrics. Section 3 describes our proposed second order method, our implementation, and is followed by an analysis of the error produced by approximations used in our methods (Subsection 3.2). In Subsection 3.3, we introduce alternate methods of solving the robust optimization problem that are implemented as a comparison to our proposed approach. Section 4 first describes the setup of a synthetic dataset along with the preliminary fairness results, and then extends the discussion to several real world datasets. We also examine the relative computational efficiency of our different methods of performing robust optimization. Lastly, Section 5 concludes the paper and discusses potential future work.

## 2. Background.

First we must discuss DNNs and some necessary notation, robust optimization, and our choice of fairness metrics.

### 2.1. Notation.

Deep neural networks (DNNs) can be represented by a parameterized mapping $f_{\boldsymbol{\theta}} : \mathcal{X} \times \Theta \to \mathcal{Y}$ from input-target pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, where $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ is the data space, $\mathcal{X} \subseteq \mathbb{R}^{n_{\text{in}}}$ is the input space, and $\mathcal{Y} \subseteq \mathbb{R}^{n_{\text{out}}}$ is the target space, and $\Theta \subset \mathbb{R}^{n_{\boldsymbol{\theta}}}$ is the parameter space. Our goal is to learn the weights $\boldsymbol{\theta} \in \Theta$ such that $f_{\boldsymbol{\theta}}(\mathbf{x}) \approx \mathbf{y}$ for all input-target pairs. Typically, learning the weights is posed as the optimization problem

$$\min_{\boldsymbol{\theta}} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) + R(\boldsymbol{\theta}) \tag{2.1}$$

where $\mathcal{T} \subset \mathcal{D}$ is the training set and $R : \Theta \to \mathbb{R}$ is a regularization term to enforce desirable properties on the weights.

For many problems with a well-chosen optimizer, we can solve (2.1) well. However, this can lead to problems such as overfitting, where the model fits the training data well but does

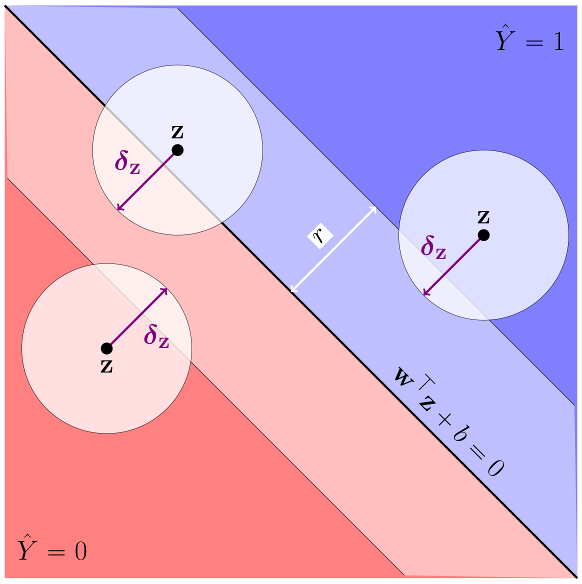


**Figure 1:** Robust optimization, visualized in the case of a linear classifier (black line) in two dimensions $\mathbf{w}^\top\mathbf{z}+b=0$. The black data points $\mathbf{z} \equiv f_{\boldsymbol{\theta}}(\mathbf{x})$ are the network outputs for various data inputs. The white circles indicate output features within a radius of $r$ of the network outputs. The direction of perturbation $\boldsymbol{\delta}_\mathbf{z}$ that maximizes the inner optimization problem is normal to the linear classifier defined by $\mathbf{w}$. Any network outputs in the white channel, $r$ away from the linear classifier, change the predicted class. Robust optimization encourages network outputs to live outside of the white channel to avoid ambiguous class predictions.

not generalize to unseen data, or a lack of robustness, where small changes to the data result in significantly different results (e.g., incorrect classifications).

**2.2. Robust Optimization.** Adversarial training promotes robustness in DNNs by introducing a perturbation $\boldsymbol{\delta}_\mathbf{x}$ for each input $\mathbf{x}$ and solving the minimax problem

$$\min_{\boldsymbol{\theta}} \quad \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} L(f_{\boldsymbol{\theta}}(\mathbf{x}+\delta_\mathbf{x}(\boldsymbol{\theta})), \mathbf{y}) + R(\boldsymbol{\theta}) \tag{2.2a}$$

$$\text{s. t.} \quad \boldsymbol{\delta}_\mathbf{x}(\boldsymbol{\theta}) \in \operatorname*{argmax}_{\|\boldsymbol{\delta}_\mathbf{x}\|_2 \le r} L(f_{\boldsymbol{\theta}}(\mathbf{x}+\boldsymbol{\delta}_\mathbf{x}), \mathbf{y}) \qquad \text{for each } (\mathbf{x},\mathbf{y}) \in \mathcal{T} \tag{2.2b}$$

We perturb the inputs $\mathbf{x}$ by $\boldsymbol{\delta}_\mathbf{x}$ and maximize the Euclidean norm of the perturbation $\|\boldsymbol{\delta}_\mathbf{x}\|_2$ (inner optimization problem (2.2b)) while optimally fitting the data (outer minimization problem (2.2a)). We build neighborhoods of radius $r$ around our training points where we can rely on our model classifying anything within the neighborhood similarly. See Figure 1 for a visualization.

The complexity of our new minimax problem is a large consideration for the applicability of our results to large scale real-world situations. To address this, we use second order information to solve the inner optimization problem efficiently in terms of computational time.

Solving this problem well means satisfying first order optimality conditions. Following [2], we first negate the loss to produce an equivalent minimization problem and set up a Lagrangian.

$$\mathcal{L}(\boldsymbol{\delta}_{\mathbf{x}}, \lambda) = -L(f_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\delta}_{\mathbf{x}}), \mathbf{y}) + \lambda(\tfrac{1}{2}\|\boldsymbol{\delta}_{\mathbf{x}}\|_2^2 - \tfrac{1}{2}r^2) \tag{2.3}$$

Here $\lambda$ is a Lagrangian multiplier and we use an equivalent version of the constraint $\frac{1}{2}\|\boldsymbol{\delta}_{\mathbf{x}}\|_2^2 \leq \frac{1}{2}r^2$ that we can differentiate more easily. The perturbation $\boldsymbol{\delta}_{\mathbf{x}}$ that maximizes the inner optimization problem of (2.2) necessarily satisfies the Karush-Kuhn-Tucker (KKT) conditions below [9].

$$\begin{aligned}
&(2.4a) && \nabla_{\boldsymbol{\delta}_{\mathbf{x}}}\mathcal{L}(\boldsymbol{\delta}_{\mathbf{x}}, \lambda) = -\nabla_{\boldsymbol{\delta}_{\mathbf{x}}}L(f_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\delta}_{\mathbf{x}}), \mathbf{y}) + \lambda\boldsymbol{\delta}_{\mathbf{x}} = \mathbf{0} && \text{(stationarity)}\\
&(2.4b) && \|\boldsymbol{\delta}_{\mathbf{x}}\|_2 \leq r && \text{(primal feasibility)}\\
&(2.4c) && \lambda \geq 0 && \text{(dual feasibility)}\\
&(2.4d) && \lambda(\|\boldsymbol{\delta}_{\mathbf{x}}\|_2 - r) = 0 && \text{(complementary slackness)}
\end{aligned}$$

Satisfying the KKT conditions ensures that gradients of the outer optimization problem are accurate; in particular, for each training sample, we have

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}L(f_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta})), \mathbf{y}) = {} & [\nabla_{\boldsymbol{\theta}'}f_{\boldsymbol{\theta}'}(\mathbf{x} + \boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta}))\nabla_f L(f_{\boldsymbol{\theta}'}(\mathbf{x} + \boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta})), \mathbf{y})]_{\boldsymbol{\theta}'=\boldsymbol{\theta}}\\
& + \left[\nabla_{\boldsymbol{\theta}'}\boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta}')\nabla_{\boldsymbol{\delta}_{\mathbf{x}}}L(f_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta}')), \mathbf{y})\right]_{\boldsymbol{\theta}'=\boldsymbol{\theta}}\end{aligned} \tag{2.5}$$

The first term in (2.5) is the traditional gradient that we want to preserve. The second term comes from considering the perturbation as a function of the network weights, $\boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta})$. From the stationarity condition (2.4a), we get that $\nabla_{\boldsymbol{\delta}_{\mathbf{x}}}L(f_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\delta}_{\mathbf{x}}), \mathbf{y})$ is parallel to $\boldsymbol{\delta}_{\mathbf{x}}$ if the perturbation is a maximizer. If the constraint is inactive ($\lambda = 0$), then $\nabla_{\boldsymbol{\delta}_{\mathbf{x}}}L(f_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\delta}_{\mathbf{x}}), \mathbf{y}) = \mathbf{0}$ and the second term is zero. If the constraint is active ($\lambda > 0$), then from primal feasibility (2.4b) we know that the perturbation must satisfy the constraint even when undergoing changes incurred from $[\nabla_{\boldsymbol{\theta}'}\boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta}')]_{\boldsymbol{\theta}'=\boldsymbol{\theta}}$. With a sufficiently small perturbation of the weights $\boldsymbol{\theta}$, the change in perturbation will follow the boundary of the constraint, nearly orthogonal to the direction of the gradient $\nabla_{\boldsymbol{\delta}_{\mathbf{x}}}L(f_{\boldsymbol{\theta}}(\mathbf{x}+\boldsymbol{\delta}_{\mathbf{x}}), \mathbf{y})$. This again makes the second term zero. Thus, if we solve the inner optimization problem well and thereby satisfy the KKT conditions, we can ignore the contribution of the second term.

**2.3. Fairness.** We use three different fairness metrics defined in [1] in our experiments. All of these fairness metrics pertain to fairness of a classifier with respect to a sensitive attribute, in terms of true labels against the classifier's predictions. In all of our experiments, the sensitive attribute $s$, true label $Y$, and classifier prediction $\hat{Y}$ are all binary. For convenience, in defining the fairness metrics, we treat $Y$ as a random variable representing an object's true label and $\hat{Y}$ as a random variable representing its prediction.

**2.3.1. Independence.** For a classifier to satisfy independence its prediction $\hat{Y}$ must be uncorrelated with the sensitive attribute $s$. This requires an equal rate of positive classifications across all sensitive groups.

$$P(\hat{Y} = 1|s = 0) = P(\hat{Y} = 1|s = 1) = P(\hat{Y} = 1) \tag{2.6}$$

For instance, if the classifier was being used to recommend hiring decisions (so $\hat{Y} = 1$ means a candidate should be hired, and $\hat{Y} = 0$ means a candidate should not), satisfying independence would mean that if the classifier hires 20% of applicants in class $s = 1$, then it also hires 20% of applicants in class $s = 0$.

### 2.3.2. Separation.
Separation is similar to independence; for separation to be satisfied $\hat{Y}$ must be conditionally independent of $s$ given the value of $Y$.

$$
\begin{aligned}
P(\hat{Y} = 1 | Y = 1, s = 0) &= P(\hat{Y} = 1 | Y = 1, s = 1) \\
P(\hat{Y} = 1 | Y = 0, s = 0) &= P(\hat{Y} = 1 | Y = 0, s = 1)
\end{aligned} \tag{2.7}
$$

Separation enforces equality of true and false positive rates. If again $\hat{Y}$ determines hiring recommendations, then $Y$ might indicate an individual's true qualifications. Separation requires that individuals with similar qualifications have an equal chance of being hired, regardless of sensitive attribute.

### 2.3.3. Sufficiency.
Sufficiency enforces the conditional independence of $Y$ and $s$ given $\hat{Y}$.

$$
\begin{aligned}
P(Y = 1 | \hat{Y} = 1, s = 0) &= P(Y = 1 | \hat{Y} = 1, s = 1) \\
P(Y = 1 | \hat{Y} = 0, s = 0) &= P(Y = 1 | \hat{Y} = 0, s = 1)
\end{aligned} \tag{2.8}
$$

Sufficiency requires that the rates of individuals with the same predicted label also having the same true label is equal across different sensitive groups. If sufficiency is satisfied, then an individual from one group who is hired by the classifier is as likely to be truly qualified as a hired individual from another group.

## 3. Our Approach.
Next we introduce our proposed second order method, and discuss its implementation. Our approach relies on approximation, so an analysis of the error produced by this approximation follows in Subsection 3.2. Then in Subsection 3.3, we introduce alternate methods of solving the robust optimization problem and their implementations to test against our proposed approach.

### 3.1. Trust Region Subproblem (TRS).
Our main algorithm (Algorithm 3.1) solves an approximation of the inner optimization problem (2.2b) using second order information. For each training sample $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}$, we fix $\boldsymbol{\theta}$ and expand the loss function using a quadratic Taylor series approximation about $\mathbf{x}$ in the direction of $\boldsymbol{\delta}_{\mathbf{x}}$.

$$
\min_{\|\boldsymbol{\delta}_{\mathbf{x}}\|_2 \le r} -L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) - (\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}))^T \boldsymbol{\delta}_{\mathbf{x}} - \tfrac{1}{2} \boldsymbol{\delta}_{\mathbf{x}}^T \nabla_{\mathbf{x}}^2 L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\, \boldsymbol{\delta}_{\mathbf{x}} \tag{3.1}
$$

To fit our constraint, we construct a Lagrangian term by squaring our initial constraint and scaling the Lagrange multiplier by one-half. This gives us a function that depends on $\boldsymbol{\delta}_{\mathbf{x}}$ and $\lambda$.

$$
\tilde{\mathcal{L}}(\boldsymbol{\delta}_{\mathbf{x}}, \lambda) = -L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) - (\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}))^T \boldsymbol{\delta}_{\mathbf{x}} - \frac{1}{2} \boldsymbol{\delta}_{\mathbf{x}}^T \nabla_{\mathbf{x}}^2 L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\, \boldsymbol{\delta}_{\mathbf{x}} + \frac{\lambda}{2}(||\boldsymbol{\delta}_{\mathbf{x}}||^2 - r^2) \tag{3.2}
$$

We approximate the optimal $\boldsymbol{\delta}_{\mathbf{x}}^*$ to the inner optimization problem as the optimal $\boldsymbol{\delta}_{\mathbf{x}}$ solution to the quadratic problem (3.1). The KKT conditions are the same as (2.4) except for the

**Algorithm 3.1** Trust Region Subproblem

```
Require: network $f_{\boldsymbol\theta} : \mathcal{X} \times \Theta \to \mathcal{Y}$, loss function $L : \mathbb{R}^{n_{\text{out}}} \times \mathcal{Y} \to \mathbb{R}$, batch $\mathcal{T}_i \subset \mathcal{T}$, trust
    region radius $r$
Ensure: Candidate perturbation per sample $\mathbf{S} \in \mathbb{R}^{n_{\text{in}} \times |\mathcal{T}_i|}$
 1: Initialize $\mathbf{S}$ as empty array
 2: for each sample $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}_i$ do
 3:     Evaluate loss and derivatives, $L(f_{\boldsymbol\theta}(\mathbf{x}), \mathbf{y})$, $\nabla_{\mathbf{x}} L(f_{\boldsymbol\theta}(\mathbf{x}), \mathbf{y})$, and $\nabla^2_{\mathbf{x}} L(f_{\boldsymbol\theta}(\mathbf{x}), \mathbf{y})$
 4:     Define perturbation as a function of Lagrange multiplier $\boldsymbol\delta_{\mathbf{x}}(\lambda)$    ▷ Equation (3.4)
 5:     Compute unconstrained perturbation $\mathbf{s}_{\mathbf{x}} = \boldsymbol\delta_{\mathbf{x}}(0)$
 6:     if $\|\mathbf{s}_{\mathbf{x}}\|_2 > r$ then
 7:         Set $\lambda_{\text{low}} = 0$, compute $\lambda_{\text{high}}$    ▷ Subsection 3.1.1
 8:         Solve for $\lambda^*$ using bisection method on the function $g(\lambda)$    ▷ Equation (3.5)
 9:         Choose optimal search direction $\mathbf{s}_{\mathbf{x}} = \boldsymbol\delta_{\mathbf{x}}(\lambda^*)$
10:     end if
11:     Concatenate $\mathbf{S}$ and $\mathbf{s}_{\mathbf{x}}$
12: end for
```

stationarity condition.

$$-\nabla_{\mathbf{x}} L(f_{\boldsymbol\theta}(\mathbf{x}), \mathbf{y}) - \nabla^2_{\mathbf{x}} L(f_{\boldsymbol\theta}(\mathbf{x}), \mathbf{y})\, \boldsymbol\delta_{\mathbf{x}} + \lambda \boldsymbol\delta_{\mathbf{x}} = \mathbf{0} \qquad \text{(stationarity)} \tag{3.3}$$

This gives us an explicit relation of $\boldsymbol\delta_x$ to $\lambda$.

$$\boldsymbol\delta_{\mathbf{x}}(\lambda) = -(\nabla^2_{\mathbf{x}} L(f_{\boldsymbol\theta}(\mathbf{x}), \mathbf{y}) - \lambda I)^{-1} \nabla_{\mathbf{x}} L(f_{\boldsymbol\theta}(\mathbf{x}), \mathbf{y}) \tag{3.4}$$

There are two cases to (3.4). If $\lambda = 0$, then the optimal $\boldsymbol\delta_{\mathbf{x}}$ for (3.1) can be found by solving a system of linear equations involving the gradient and the Hessian. Alternatively, if $\lambda \neq 0$, then complementary slackness enforces $\|\boldsymbol\delta_{\mathbf{x}}\|_2 = r$, so we need to find $\lambda$ such that $\|\boldsymbol\delta_{\mathbf{x}}(\lambda)\|_2 = r$.

We note that Algorithm 3.1 and our derivation uses a "per-sample" option. This means the trust region constraint is applied independently for each sample in the input dataset; i.e., for each data point, we solve a separate trust region optimization problem and perturb. This is beneficial when different data points require different level of adjustment. An alternative approach would be to use a "global" option; i.e., a single constraint is applied to the entire batch of data. The "per-sample" approach is beneficial because we can optimize the adversarial perturbation for each samples (i.e., increase robustness) and offers the potential to use a different trust region radius per sample (we have not programmed this adaptability into our code yet). The trade off is that the "per-sample" method is run sequentially over the batch samples, which can be slow. There is a potential for parallelization; however, this is non-trivial to code, particularly when ensuring the gradients track properly for automatic differentiation. We consider as a future improvement of the repository. During our experiments, we had a choice to use either but we mainly used "per-sample" which is why it was included.

**3.1.1. The Bisection Method Bracket.** To find a value for $\lambda$ such that $\|\boldsymbol\delta_{\mathbf{x}}(\lambda)\|_2 = r$, we build a univariate function $g(\lambda) := \|\boldsymbol\delta_{\mathbf{x}}(\lambda)\|_2 - r$ and find a root of this function. Applying some linear algebra to (3.4), one can show that

$$g(\lambda) = \|\boldsymbol{\delta}_{\mathbf{x}}(\lambda)\|_2 - r \tag{3.5a}$$
$$= \| - (\nabla^2_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) - \lambda I)^{-1} \nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 - r \tag{3.5b}$$
$$= \|(QDQ^\top - \lambda I)^{-1} \nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 - r \tag{3.5c}$$
$$= \|Q(D - \lambda I)^{-1} Q^\top \nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 - r \tag{3.5d}$$
$$= \|(D - \lambda I)^{-1} Q^\top \nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 - r \tag{3.5e}$$

where $\nabla^2_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) = QDQ^T$ is the eigendecomposition of the Hessian. Because the Hessian is symmetric, by the Spectral Theorem, we know $Q$ is orthogonal and $D$ is diagonal and real-valued.

We use the bisection method [5] to find a root of $g(\lambda) = 0$. We do recognize that it may not seem appropriate to use a bisection method here because $g$ is not continuous at eigenvalues of the Hessian of $L$. However, given the complexity of the function $g$ and of robust optimization in general, we wanted to use a straight-forward approach to find a root of $g$. In practice, we did not seem to run into any numerical issues in our code using the bisection method. Moreover, we obtained the same final fairness and accuracy results using a first order projected gradient descent (PGD) method, outlined later, as using a bisection method with our new second-order optimization approach. Considering these things, we considered the bisection method to be adequate for the purposes of this work, and we leave it for future work to improve the root-finding strategy of our optimization method.

In order to use the bisection method, we need to establish a bracket $[\lambda_{\text{low}}, \lambda_{\text{high}}]$ such that $g$ has different signs at the endpoints; that is, $g(\lambda_{\text{low}})g(\lambda_{\text{high}}) < 0$. If $\lambda = 0$, then constraint is satisfied. Thus, $g(0) \geq 0$ and, in practice, positive, so $\lambda_{\text{low}} = 0$ is a good candidate. To find the upper bound, we first bound the norm

$$\|(D - \lambda I)^{-1} Q^\top \nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 \leq \|(D - \lambda I)^{-1}\|_2 \|\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 \tag{3.6}$$

$$= \sqrt{\sum_{i=1}^{n_{\text{in}}} \frac{1}{(d_i - \lambda)^2}} \|\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 \tag{3.7}$$

Following from [9], as $\lambda \to d^+_{\max}$, the upper bound approaches $+\infty$ and as $\lambda \to \infty$, the upper bound approaches 0. This guarantees that there is some $\lambda \in (d_{\max}, \infty)$ such that the upper bound is less than $r$. If we let

$$\lambda_{\text{high}} = |d_{\max}| + \frac{\sqrt{n}\|\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2}{r} \tag{3.8}$$

then $(d_i - \lambda_{\text{high}})^2 \geq \dfrac{n\|\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2^2}{r^2}$ for $i = 1, \ldots, n_{\text{in}}$. Substituting into the upper bound, we get

$$\sqrt{\sum_{i=1}^{n_{\text{in}}} \frac{1}{(d_i - \lambda)^2}} \|\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 \leq \frac{\sqrt{n} r}{\sqrt{n}\|\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2} \|\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\|_2 = r. \tag{3.9}$$

Thus, we have found a bracket for the bisection method.

**3.2. Algorithm Analysis.** When solving the inner optimization problem using a second order approximation, we would like to know how well this approximation actually solves this problem. For specific classes of models, loss functions, and activation functions, we can confine the error explicitly to depend on high orders of the perturbation $\boldsymbol{\delta}_\mathbf{x}$ and loss function derivatives.

**3.2.1. Affine model.** An affine model $f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ with weight vector $\mathbf{w}$ and a scalar bias $b$ combined with a logistic regression loss function $L$ and a sigmoid activation function $\sigma$ is convex with respect to inputs. The loss is given explicitly as

$$L(f_{\mathbf{w},b}(\mathbf{x}), y) = -y \ln[\sigma(\mathbf{w}^\top \mathbf{x} + b)] - (1-y)\ln[1 - \sigma(\mathbf{w}^\top \mathbf{x} + b)] \tag{3.10}$$

with $\sigma(z) = (1+e^{-z})^{-1}$. Introducing the perturbation $\boldsymbol{\delta}_\mathbf{x}$ results in a specific case of the inner optimization problem in (2.2b). To solve this optimization problem without approximation, we introduce as before a Lagrange multiplier $\lambda$ with the constraint $\|\boldsymbol{\delta}_\mathbf{x}\|^2 - r^2 \leq 0$.

$$\begin{aligned}\mathcal{L}_{\text{aff}}(\boldsymbol{\delta}_\mathbf{x}, \lambda) = -y \ln(\sigma(\mathbf{w}^\top(\mathbf{x} + \boldsymbol{\delta}_\mathbf{x}) + b)) - (1-y)\ln(1 - \sigma(\mathbf{w}^\top(\mathbf{x} + \boldsymbol{\delta}_\mathbf{x}) + b))\\ + \frac{1}{2}\lambda(\|\boldsymbol{\delta}_\mathbf{x}\|_2^2 - r^2)\end{aligned} \tag{3.11}$$

The first order optimality conditions for (3.11) tell us that at the optimal $\boldsymbol{\delta}_\mathbf{x}$,

$$\nabla_{\boldsymbol{\delta}_\mathbf{x}} \mathcal{L}_{\text{aff}}(\boldsymbol{\delta}_\mathbf{x}, \lambda) = (-y + \sigma(\mathbf{w}^\top(\mathbf{x} + \boldsymbol{\delta}_\mathbf{x}) + b))\mathbf{w} + \lambda \boldsymbol{\delta}_\mathbf{x} = \mathbf{0}. \tag{3.12}$$

To compare the exact solution from equation (3.12) to the approximation made when solving using the trust region method of section 3.1, we find the second order approximation of the loss function $L(f_{\mathbf{w},b}(\mathbf{x}), y)$ by a Taylor expansion in $\mathbf{x}$ in the direction of $\boldsymbol{\delta}_\mathbf{x}$.

$$L(f_{\mathbf{w},b}(\mathbf{x} + \boldsymbol{\delta}_\mathbf{x}), y) \approx L(f_{\mathbf{w},b}(\mathbf{x}), y) + \nabla_\mathbf{x}^\top L(f_{\mathbf{w},b}(\mathbf{x}), y)\boldsymbol{\delta}_\mathbf{x} + \frac{1}{2}\boldsymbol{\delta}_\mathbf{x}^\top \nabla_\mathbf{x}^2 L(f_{\mathbf{w},b}(\mathbf{x}), y)\boldsymbol{\delta}_\mathbf{x} \tag{3.13}$$

where the gradient and Hessian are

$$\begin{aligned}\nabla_\mathbf{x} L(f_{\mathbf{w},b}(\mathbf{x}), y) &= (-y + \sigma(\mathbf{w}^\top \mathbf{x} + b))\mathbf{w}\\ \nabla_\mathbf{x}^2 L(f_{\mathbf{w},b}(\mathbf{x}), y) &= \mathbf{w}\sigma'(\mathbf{w}^\top \mathbf{x} + b)\mathbf{w}^\top.\end{aligned} \tag{3.14}$$

The associated Lagrangian is

$$\tilde{\mathcal{L}}_{\text{aff}}(\boldsymbol{\delta}_\mathbf{x}, \lambda) = L(f_{\mathbf{w},b}(\mathbf{x}), y) + \nabla_\mathbf{x}^\top L(f_{\mathbf{w},b}(\mathbf{x}), y)\boldsymbol{\delta}_\mathbf{x} + \frac{1}{2}\boldsymbol{\delta}_\mathbf{x}^\top \nabla_\mathbf{x}^2 L(f_{\mathbf{w},b}(\mathbf{x}), y)\boldsymbol{\delta}_\mathbf{x} + \frac{1}{2}\lambda(\|\boldsymbol{\delta}_\mathbf{x}\|^2 - r^2). \tag{3.15}$$

As before, take the gradient with respect to $\boldsymbol{\delta}_\mathbf{x}$ and set it equal to zero to solve using first-order optimization conditions.

$$\nabla_{\boldsymbol{\delta}_\mathbf{x}} \mathcal{L}(\boldsymbol{\delta}_\mathbf{x}, \lambda) = (-y + \sigma(\mathbf{w}^\top \mathbf{x} + b))\mathbf{w} + \mathbf{w}\sigma'(\mathbf{w}^\top \mathbf{x} + b)\mathbf{w}^\top \boldsymbol{\delta}_\mathbf{x} + \lambda \boldsymbol{\delta}_\mathbf{x} = \mathbf{0} \tag{3.16}$$

Comparing (3.12) (LHS) and (3.16) (RHS), the discrepancy between the exact solution and the approximation is restricted to

$$\sigma(\mathbf{w}^\top(\mathbf{x} + \boldsymbol{\delta}_\mathbf{x}) + b) \neq \sigma(\mathbf{w}^\top \mathbf{x} + b) + \sigma'(\mathbf{w}^\top \mathbf{x} + b)\mathbf{w}^\top \boldsymbol{\delta}_\mathbf{x}. \tag{3.17}$$

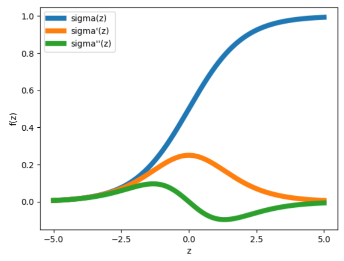


**Figure 2:** Flattening behavior of sigmoidal function, $\sigma$, derivatives.

Now, appling a Taylor expansion centered at $\mathbf{x}$ in the ball $\mathbf{x}+\boldsymbol{\delta}_{\mathbf{x}}$ to the LHS, we obtain:

$$\sigma(\mathbf{w}^\top(\mathbf{x}+\boldsymbol{\delta}_{\mathbf{x}})+b) \approx \sigma(\mathbf{w}^\top\mathbf{x}+b)+\sigma'(\mathbf{w}^\top\mathbf{x}+b)\mathbf{w}^\top\boldsymbol{\delta}_{\mathbf{x}}+\frac{1}{2}\boldsymbol{\delta}_{\mathbf{x}}^\top\mathbf{w}\sigma''(\mathbf{w}^\top\mathbf{x}+b)\mathbf{w}^\top\boldsymbol{\delta}_{\mathbf{x}} \tag{3.18}$$

We have recovered the RHS of (3.17), so the error is due to the truncation of the second-order and higher terms of (3.18). In this case with a sigmoidal loss function, that means this error depends on the magnitude of $|\sigma''(z)|$ and the $\boldsymbol{\delta}_{\mathbf{x}}$ for which we solved.

For any sigmoidal function, their structure gives first and second order derivatives of the classes shown in Figure 2. For our sigmoid function defined as $\sigma(z) = (1+e^{-z})^{-1}$ with $|\sigma''(z)| \leq 0.1$, and in general for any choice of sigmoidal function this flattening of higher-order derivatives will be observed. By nature $\boldsymbol{\delta}_{\mathbf{x}}$ is bounded by the data since it defines the perturbation from a given point, and a perturbation larger than the size of the data space in any given dimension would be meaningless. For data in the form of continuous numerical values normalized to be between 0 and 1, as in our case, each component of $\boldsymbol{\delta}_{\mathbf{x}}$ must be less than 1. In practice, $\boldsymbol{\delta}_{\mathbf{x}}$ tends to be much smaller than that. This bound means higher order terms are generally quite small, and for this combination of loss function, activation function, model, and radii on the order of $10^{-1}$, the approximation error is on the order of $\|\boldsymbol{\delta}_{\mathbf{x}}\|^2|\sigma''(z)| \approx 10^{-3}$.

## 3.3. Other Methods.

We compare our trust region subproblem (TRS) algorithm to random perturbation to examine whether or not it is important to solve the inner optimization problem well. We also use random perturbation, along with a projected gradient descent (PGD) method, to verify that our second order TRS approach has greater computational efficiency than only using lower order information.

### 3.3.1. Random Perturbation.

For each data point, we sample the perturbation $\boldsymbol{\delta}_x$ randomly from a multivariate standard normal distribution and rescale to the length of the trust region radius. This method acts as a control in our experiments to show the advantages of solving for an optimal perturbation.

### 3.3.2. Projected Gradient Descent (PGD).

In order to test that our second order TRS approach is computationally faster than using purely first order information, we also implemented a version of gradient descent for our inner optimization problem. Since our inner problem has the constraint $\|\boldsymbol{\delta}_x\|_2 \leq r$, we cannot use vanilla gradient descent, and instead use projected gradient descent (PGD) [3, 10]. PGD operates similarly to standard gradient descent, but once it has found its optimal step it projects the step onto the constrained set before returning it. Mathematically, this looks like:

$$\boldsymbol{\delta}_{\boldsymbol{x}}^{(k+1)} = P\left[\boldsymbol{\delta}_{\boldsymbol{x}}^{(k)} + \alpha^{(k)} \cdot \nabla_{\boldsymbol{x}} L(f_{\boldsymbol{\theta}}(\boldsymbol{x} + \boldsymbol{\delta}_{\boldsymbol{x}}), \boldsymbol{y})\right] \tag{3.19}$$

where $\boldsymbol{\delta}_{\boldsymbol{x}}^{(k)}$ is the $k$th iterate, $\alpha^{(k)}$ is the step size at the $k$th iteration, and P is the projection operator $P(\mathbf{x}) = \operatorname{argmin}_{\mathbf{z}:||\mathbf{z}||_2 \le r} ||\mathbf{x} - \mathbf{z}||_2^2$. This projection operator turns out to be very simple, returning the inputted point if the point already satisfies the constraint, or scaling the point inward to the boundary of the constraint if it is outside it. In particular,

$$P(x) = \min\left\{1, \frac{r}{||\mathbf{x}||_2}\right\}\mathbf{x}. \tag{3.20}$$

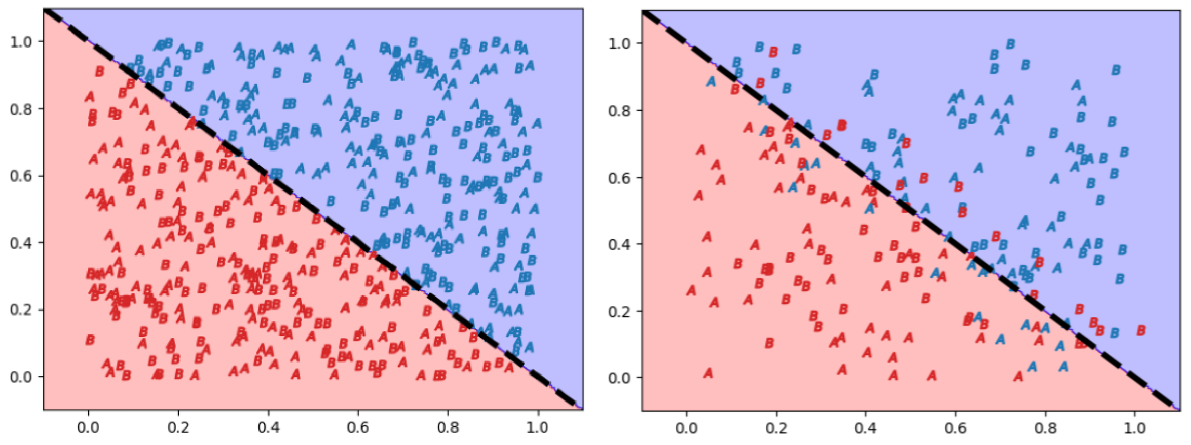
Numerical experiments on how PGD performs with adversarial training [13, 6] have shown that PGD is a reliable method when it comes to solving robust optimization problems.

## 4. Numerical Results.

Now we present results of our numerical experiments pertaining to fairness, accuracy, and computational time. Subsections 4.1, 4.2, and 4.3 discuss fairness and accuracy results on three datasets. For each dataset, we compute the fairness and accuracy results for nonrobust training, for robust training with various radii, and random perturbations. We measure the absolute difference across sensitive attributes for analysis of each fairness metric, and the closer this difference is to zero, the fairer the classifier is with respect to that metric. Subsection 4.4 presents our results on relative computational time across our various methods for solving the inner optimization problem.

### 4.1. Synthetic Data (Unfair2D).

The primary dataset we used for carrying out numerical experiments was a synthetic dataset. Individuals belonging to two different groups, labeled with respect to a sensitive attribute $A$ or $B$, are being hired on the basis of two numeric scores $x_1$ and $x_2$. Individuals have a binary label that is either “should be hired” or “shouldn’t be hired,” and we train a linear classifier to decide whether or not to hire an individual. The data is initially fair (Figure 3a), and we introduce unfair bias into the data by artificially raising the scores of all $B$s while lowering the scores of all $A$s (Figure 3b). In the real world, this could be a manifestation of structural unfairness in which $B$s are more likely to belong to a wealthy socioeconomic class, and thus can afford training that boosts their scores, whereas $A$s do not have this opportunity. In fact, $A$s may not only lack the advantage of $B$s, but also have an active disadvantage, such as an increased likelihood of needing to work longer hours, impeding time for study and test prep, lowering their scores.



**(a)** Pre-shift data **(b)** Sample of post-shift data

**Figure 3:** Points are colored based on their original location in the blue region ($Y = 1$) or red region ($Y = 0$). Post shift, note the unfair presence of red $B$s in the blue region and blue $A$s in the red region.

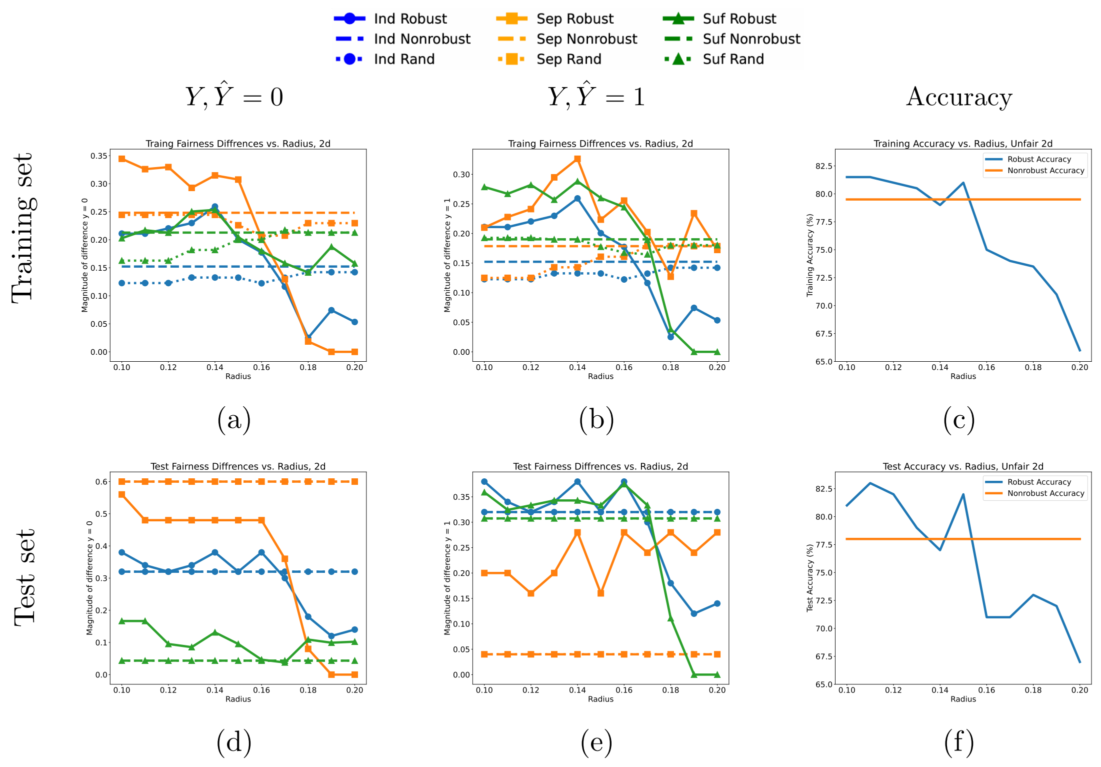
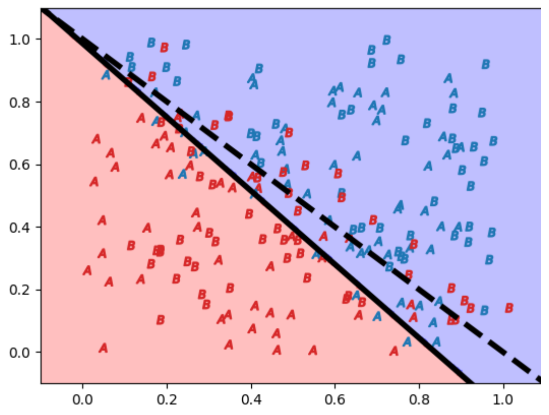
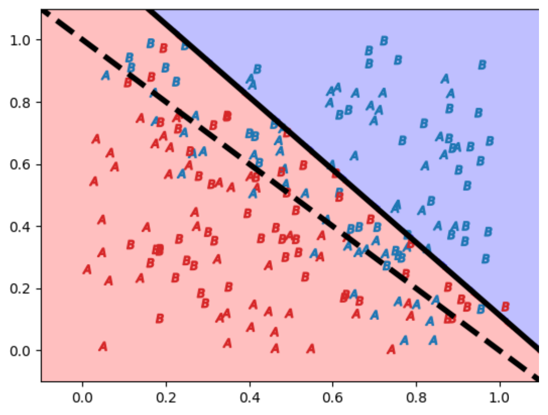


**Figure 4:** Synthetic fairness and accuracy results. For fairness, values closer to zero are desirable.



**(a)** Nonrobust Classifier



**(b)** Robust Classifier (r=0.18)

| **Diff:** | $Y = 0, \|S1 - S0\|$ | $Y = 1, \|S1 - S0\|$ |
|---|---|---|
| Ind. | 0.152 | 0.152 |
| Sep. | 0.248 | 0.179 |
| Suff. | 0.213 | 0.190 |

**Training Accuracy:** 79.5%
**Test Accuracy:** 78.0%

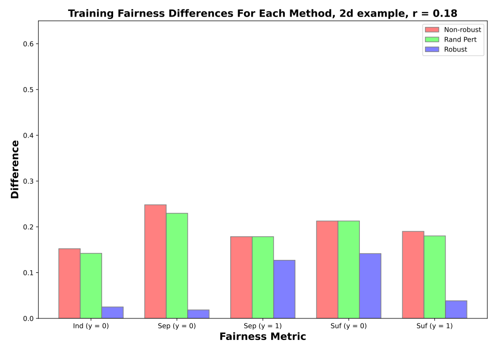
| **Diff:** | $Y = 0, \|S1 - S0\|$ | $Y = 1, \|S1 - S0\|$ |
|---|---|---|
| Ind. | 0.025 | 0.025 |
| Sep. | 0.019 | 0.127 |
| Suff. | 0.142 | 0.038 |

**Training Accuracy:** 73.5%
**Test Accuracy:** 73.0%

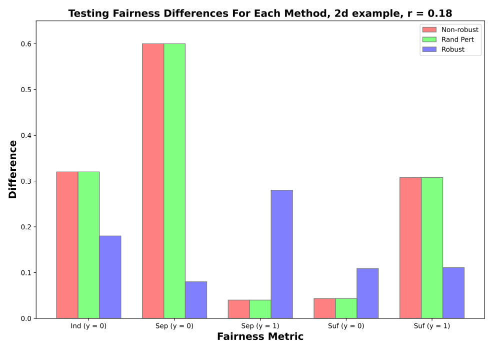
**Figure 5:** Comparative Analysis of Non-Robust and Robust Classifiers

We compare nonrobust training on this synthetic dataset with robust training (using the TRS method) and random perturbation for 11 different perturbation radii, with the radius increasing in 0.01 increments from 0.1 up to 0.2. These experiments were run with a total of 10 training epochs and a learning rate of 0.01 in the outer optimization problem. Four plots of fairness metric differences versus radius are shown in Figure 4, as well as plots of the training and testing accuracy versus radius. For many radii in the lower end of the plotted range, many of the fairness differences are worse in the training data with robust training than with nonrobust. However, some fairness improvement can be seen with robust training.

In all cases, at least two of the three fairness metrics show a downward trend for robust training. This suggests that while robust training may worsen fairness for a very small radius, fairness improvement is possible at more appropriate radii. At a radius of 0.18, all of the robust fairness differences are better than the corresponding nonrobust ones in the training data, although it does lead to a decrease in test accuracy from around 78% to 73% (Figure 5). The nonrobust classifier is visualized in Figure 5a versus the robust classifier in Figure 5b. In Figure 5b, robust optimization is improving fairness by raising the bar, giving a positive classification to only the most qualified individuals. It eliminates nearly all of the false positive $B$s that exist with the original dashed classifier and greatly increases the quantity of blue $B$s in the red region, equalizing false negative rates. Increasing the radius even further to 0.2 with 20 epochs of training, the robust classifier eventually classifies nothing positively. Our robust classifier is not helping the disadvantaged $A$s in the process of improving fairness – it



**(a)** Training fairness differences



**(b)** Test fairness differences

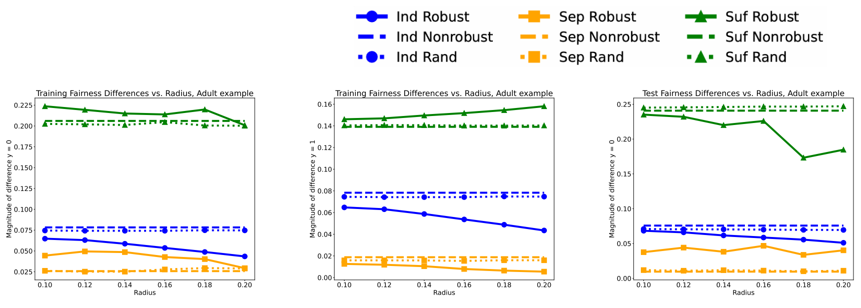
**Figure 6:** Fairness differences (r=0.18). Left bar is nonrobust, middle is random, right is robust.

is only hurting the advantaged $B$s. While this does not provide an indication of how robust training would work on every dataset, it does illustrate that robust optimization may improve fairness in an unexpected or unintended way.

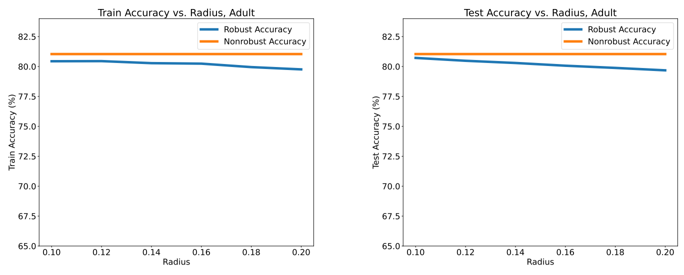
There is an advantage to solving the inner optimization problem well instead of just using random perturbations. In Figure 4, the fairness differences for random perturbations are either the same or stay close to the nonrobust differences. This stands in contrast to robust training, where fairness differences are initially high and then get significantly lower, surpassing nonrobust differences. Figure 6 also exhibits this advantage.

**4.2. Adult Dataset.** We also extended our numerical experiments to real-world datasets. The Adult dataset [11] consists of demographic data about individuals that are used to classify whether their annual income is more than $\$50,000$. Note that the dataset predominantly consists of white males in the age range of 25-60. It contains 48,842 instances and each instance is described using 15 attributes. We want to look at the 5 continuous numerical attributes (age, education-num, capital-gain, capital-loss, hours-per-week) for analysis. The income (salary) data is converted into binary form (1 for $\leq 50k$, 0 for $> 50K$), and the protected attribute can be sex or race.

Unlike our synthetic data, the adult example yielded mixed results in terms of fairness improvement. For the training data, Figure 7a and Figure 7b show that only three out of the six differences were measured to be better with robust training. There was a similar result for the test data as seen in Figure 7c and Figure 7d. Despite only having a 50% improvement rate, the majority of the fairness metrics exhibit a downward trend, and when there is an improvement robust optimization outperforms random perturbation significantly. The expected accuracy-robustness trade-off is present (Figure 7e), with both the training and test robust accuracy decreasing with increasing radii. However, unlike in the synthetic dataset, the decay appears to be linear and does not spike at certain radii, and does not yield



**(a)** **(b)** **(c)** **(d)**



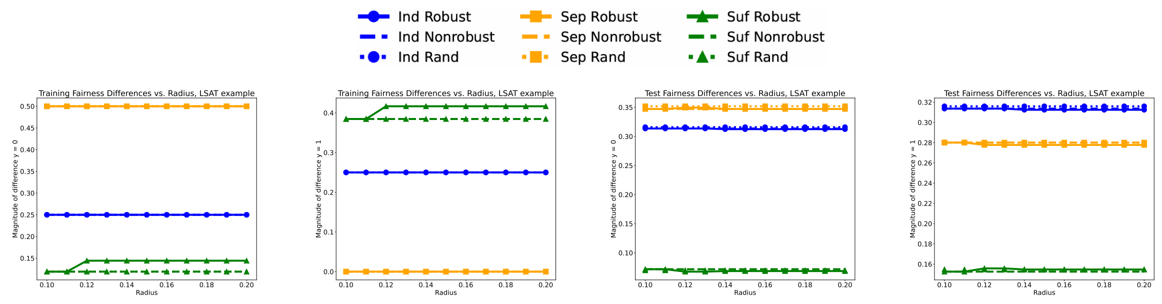
**(e)** Adult robust training accuracy.

**Figure 7:** Fairness ((a)-(d)) and accuracy (e) trends in the adult dataset for nonrobust and robust training.

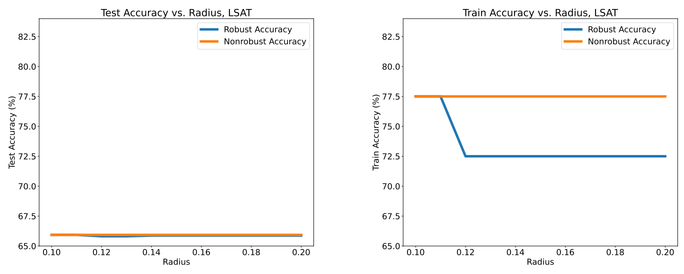
better accuracy for smaller radii. Overall, there is still reasonable case for improving fairness metrics at the expense of classifier accuracy.

**4.3. LSAT Data.** Another extension to a real-world dataset comes from the Law School Admissions Council (LSAC) [12]. This dataset was collected to explore the reasons behind low bar passage rates among racial and ethnic minorities. We train our classifier to predict whether or not a student will pass the bar, based on their Law School Admission Test (LSAT) score and undergraduate GPA. We are using GPA and LSAT scores because they are the strongest predictors for passing the bar examination. Our primary interest lies in examining five key features of the dataset: the bar exam pass/fail prediction made by a DNN, the gender of the student, their LSAT score, the true bar exam pass/fail value for the student, and their race. For the purpose of our experiment, the race feature is made binary to indicate a student as either white or non-white, which is used as the sensitive attribute.

Unlike the two previous examples, there is not a lot of fluctuation with LSAT robust results (Figure 8). Surprisingly, robust optimization seems not to deviate from nonrobust training. As we have seen, using a random perturbation yields fairness results that are very similar to the results of non-robust training. For this dataset, robust optimization also performs very similarly to just picking a random perturbation. This is especially the case with



**(a)** **(b)** **(c)** **(d)**



**(e)** LSAT robust training accuracy.

**Figure 8:** Fairness ((a)-(d)) and accuracy (e) trends in the LSAT dataset for nonrobust and robust training.

the independence and separation metrics. The only two notable deviations come from the sufficiency metric in Figure 8a and Figure 8b of the training dataset. This might be attributable to our definion of the sensitive attribute as white vs nonwhite, which creates a very small dataset due to the dominance of white individuals in the original dataset. For this specific example, robust optimization does not improve fairness, and in fact performs worse since it loses accuracy when yielding the same fairness results.

**4.4. Efficiency Comparison.** We gathered time data to see if the TRS method converged faster than PGD on our datasets. On each of the three datasets above - synthetic, Adult, and LSAT - we computed for each radius the average epoch time elapsed for TRS, PGD, and random perturbation. To compare the speed of the TRS method and PGD, we examine the ratio of the average PGD epoch time to the average TRS epoch time, looking at the extreme values of this ratio to get a range of how much faster the trust region subproblem was than PGD across all radii. The results are shown in Table 1.

Random perturbation is the fastest adversarial training method in all three datasets. This is expected, as it does not actually solve the optimization problem; its only computation task is generating a random vector and rescaling it. It is also noteworthy that using the TRS method consistently is computationally faster than using PGD. Over all radii shown, training

**Table 1:** Average Epoch Times. For each dataset, the first three rows show the average epoch times for each of the three robust optimization approaches, where a lower value indicates faster computational performance. The fourth row shows the ratio of PGD to TRS time, where a ratio greater than 1 indicates that the TRS approach was faster than the PGD approach for the given radius. The gray values highlight the minimum ratio of PGD to TRS time over all radii, while the yellow values highlight the maximum ratio.

| | **Radii** | .10 | .12 | .14 | .16 | .18 | .20 |
|---|---|---|---|---|---|---|---|
| **Synthetic** | **PGD** | 2.680 | 3.597 | 4.228 | 4.486 | 5.061 | 5.080 |
| | **TRS** | 1.945 | 1.875 | 1.806 | 1.891 | 1.742 | 1.752 |
| | **RND** | 0.0387 | 0.0366 | 0.0387 | 0.0389 | 0.0387 | 0.0375 |
| | **PGD/TRS** | 1.377 | 1.919 | 2.340 | 2.373 | 2.904 | 2.900 |
| **Adult** | **PGD** | 512.970 | 593.173 | 697.399 | 1146.856 | 1689.761 | 1854.932 |
| | **TRS** | 60.372 | 61.557 | 57.723 | 58.707 | 57.492 | 59.061 |
| | **RND** | 0.0947 | 0.101 | 0.0972 | 0.0919 | 0.0974 | 0.0939 |
| | **PGD/TRS** | 8.497 | 9.636 | 12.082 | 19.535 | 29.391 | 31.407 |
| **LSAT** | **PGD** | 1.129 | 1.559 | 2.844 | 3.575 | 3.347 | 2.752 |
| | **TRS** | 0.396 | 0.396 | 0.421 | 0.371 | 0.414 | 0.385 |
| | **RND** | 0.0107 | 0.0123 | 0.0154 | 0.0122 | 0.0162 | 0.0104 |
| | **PGD/TRS** | 2.852 | 3.936 | 6.762 | 9.639 | 8.094 | 7.150 |

with TRS is between 1.4 and 2.9 times faster than PGD in the synthetic dataset, between 8.5 and 31.4 times faster in Adult, and between 2.9 and 9.6 times faster in LSAT. The very short average epoch times for the LSAT dataset are due to the significantly smaller scale of the input data. All of the smallest factors of time improvement of TRS relative to PGD (highlighted in gray) are greater than 1 suggesting that the trust region subproblem has a consistent advantage over PGD in computational speed.

Looking at the PGD/TRS ratios, the factor of improvement that TRS has in computational time over PGD appears to be higher in the real-world datasets than in the synthetic dataset. The real-world datasets, and especially Adult, are trained on larger amounts of data, so the advantage of TRS over PGD seems to scale with the size of the dataset. This advantage of the trust region subproblem also improves with larger perturbation radii. In particular, the minimum factor of improvement (gray) always occurs with the smallest radius, and the maximum factor of improvement (yellow) always occurs with one of the three largest radii.

**5. Conclusion.** In our affine linear model setup, we were able to see improvement in fairness by using robust optimization. In the synthetic dataset, whenever there was an improvement, the gain was a significant reduction in fairness difference magnitudes (which are ideally zero). In our numerical experiments extending to real-world datasets, we have shown that robust training performs similarly to non-robust training even in the worst-case scenario (LSAT dataset). Across all three datasets, the accuracy of robust optimization decreased as the radius increased, the majority of the fairness metrics displayed a downward trend as the perturbation radius increased, and when fairness improved with robust training, precise solutions to the inner optimization problem outperformed randomly selected solutions. Furthermore, we were able to quantify the fact that, with the help of `hessQuik`, using second-order information is much faster for solving our class of optimization problem.

We acknowledge that while we were able to achieve positive results with our experiments in both synthetic and real-world datasets, there are a few mathematical limitations to our results that prevent generalization to higher-dimensional applications. We used a neural network in our training with only one hidden layer, our experiments were conducted using a linear and binary classifier, and our sensitive attribute was binary. This motivates future exploration of extending our approach to deeper neural networks, multinomial classification, and other fairness metrics relevant to those cases. It may help to improve fairness even further to introduce a regularization term to our approach to penalize violations of our fairness metrics, which is another avenue for further work. There limitations of our implementation. For PGD, we used an arbitrary step size instead of varying the step size as training proceeds. Additionally, we did not solve our inner optimization problems in parallel. Parallelizing the computations for our inner optimization problem could provide a significant reduction in overall computation time.

Despite these limitations, this work demonstrates initial promise for the ability of robust training to bring about fairness improvement in machine learning models, and motivates further research on similar methodologies.

**Acknowledgments.** This work was supported by NSF award DMS-2051019 and was completed during the "Computational Mathematics for Data Science" REU/RET program in the summer of 2023. We would like to thank Dr. Elizabeth Newman, our mentor, and the rest of the faculty who participated in the program for their feedback and support.

## REFERENCES

[1] S. Barocas, M. Hardt, and A. Narayanan, *Fairness and Machine Learning: Limitations and Opportunities*, fairmlbook.org, 2019. http://www.fairmlbook.org.

[2] A. Beck, *Introduction to Nonlinear Optimization*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2014, https://doi.org/10.1137/1.9781611973655, https://epubs.siam.org/doi/abs/10.1137/1.9781611973655, https://arxiv.org/abs/https://epubs.siam.org/doi/pdf/10.1137/1.9781611973655.

[3] A. Beck, *First-order methods in optimization*, SIAM, 2017.

[4] N. Furl, P. Phillips, and A. J. O'Toole, *Face recognition algorithms and the other-race effect: computational mechanisms for a developmental contact hypothesis*, Cognitive Science, 26 (2002), pp. 797–815, https://doi.org/https://doi.org/10.1016/S0364-0213(02)00084-8, https://www.sciencedirect.com/science/article/pii/S0364021302000848.

[5] A. Kaw, *Numerical methods with applications (kaw)*, University of South Florida, 2011, https://math.libretexts.org/Under_Construction/Numerical_Methods_with_Applications_(Kaw).

[6] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, *Towards deep learning models resistant to adversarial attacks*, in International Conference on Learning Representations, 2018, https://openreview.net/forum?id=rJzIBfZAb.

[7] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, *A survey on bias and fairness in machine learning*, CoRR, abs/1908.09635 (2019), http://arxiv.org/abs/1908.09635, https://arxiv.org/abs/1908.09635.

[8] E. Newman and L. Ruthotto, *'hessquik': Fast hessian computation of composite functions*, Journal of Open Source Software, 7 (2022), p. 4171, https://doi.org/10.21105/joss.04171, https://doi.org/10.

21105/joss.04171.
[9] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, NY, USA, 2e ed., 2006.
[10] N. Parikh, S. Boyd, et al., *Proximal algorithms*, Foundations and trends® in Optimization, 1 (2014), pp. 127–239.
[11] T. L. Quy, A. Roy, V. Iosifidis, W. Zhang, and E. Ntoutsi, *A survey on datasets for fairness-aware machine learning*, WIREs Data Mining and Knowledge Discovery, 12 (2022), https://doi.org/10.1002/widm.1452, https://doi.org/10.1002%2Fwidm.1452.
[12] L. F. Wightman, *Lsac national longitudinal bar passage study. lsac research report series.*, 1998, https://api.semanticscholar.org/CorpusID:151073942.
[13] H. Xu, X. Liu, Y. Li, A. K. Jain, and J. Tang, *To be robust or to be fair: Towards fairness in adversarial training*, 2021, https://arxiv.org/abs/2010.06121.