Fast & Fair: Efficient Second-Order Robust Optimization for Fairness in ML*

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

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 to improve fairness in DNNs, and leveraging second-order information, we are able to efficiently find a solution.

Key words. example, LaTeX

MSC codes. 68Q25, 68R10, 68U05

1. Introduction.

Machine learning has become an integral part of data analysis and powerful algorithms that can reveal underlying patterns and structures in data. Deep Neural Networks (DNNs) in particular are known to be very good at classifying complex data; however, we are concerned by the tendency of DNNs to inherit bias from the datasets they operate on. Bias in this sense is not statistical bias, but the ways in which individuals are advantaged or disadvantaged unfairly based on inherent characteristics, and the ways this can manifest in data. This can be especially problematic in areas where machine learning is used to make life-altering decisions such as criminal justice [2] and corporate hiring [4].

The widespread, ever expanding use of machine learning poses a significant ethical question when models are known to perpetuate societal biases [4]. 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 to do what we can 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 classifying differently small variations in data, known as adversarial attacks, while still optimally fitting it. One drawback of adversarial training is that it can introduce differences in accuracy and robustness between different groups of data [8]. The Fair-Robust-Learning framework was proposed to reduce this unfairness problem in adversarial training [8] Annie: do we need to cite this twice in a row?. Our research shares the goal of addressing fairness issues in DNNs through the use of adversarial training techniques. However, we intend to explore the effects

*Submitted to the editors DATE.

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)

Advised by Dr. Elizabeth Newman¶

of adversarial training on fairness generally, without a major focus on catering to specific types of data. The unique aspect of our research is our work on making the solution of the robust optimization problem more efficient. To this end, we explore the use of second-order information to accelerate training, a concept that was not addressed in previous work [8].

In our paper, we provide an analysis of our proposed method, the effectiveness of adversarial training for fairness improvement, and the efficiency of utilizing second-order information to solve optimization problems. The main extensions and contributions of our work are the following:

- **Trust region subproblem:** A second-order method for solving the inner optimization problem. We describe the setup of this method and illustrate its effectiveness.
- **Robust optimization improves fairness:** We used three different optimizers to run experiments on four different datasets and compared the fairness results between the three datasets. The first dataset is synthetic and the other two are real-world datasets of contrasting sizes.
- **Faster than PGD:** Leveraging HessQuik [5] to efficiently find exact Hessians (as opposed to using automatic differentiation) improved the time it takes to solve for an optimal perturbation radius.
- **Publishing code:** Our python implementation and experiments are available at https://github.com/elizabethnewman/fast-n-fair/tree/main.
- Annie: bullet points need reworking

The paper is organized as follows: section 2 introduces DNNs and the necessary notations, robust optimization, and our choice of fairness metrics. Section 3 describes our proposed second order method, details about the implementation of a solution, and is followed by an analysis of the error produced by approximations used in our methods (3.2). In section 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 example along with the preliminary fairness results, and then extends the discussion to several real world datasets. We also examine the efficiency between the different robust optimization methods. Lastly, section 5 concludes the paper and discusses potential future work.

2. Background.

2.1. Notation. Annie: todo

Deep neural networks (DNNs) are parameterized mappings $f_{\boldsymbol{\theta}} : \mathcal{X} \times \mathbb{R}^{n_{\boldsymbol{\theta}}} \to \mathcal{Y}$ from input-target pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, where data space $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$, input space $\mathcal{X} \subseteq \mathbb{R}^{n_{\text{in}}}$, and target space $\mathcal{Y} \subseteq \mathbb{R}^{n_{\text{out}}}$. Our goal is to learn the weights $\boldsymbol{\theta}$ 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 data and a subsample of the data space $\mathcal{D}$ and $R : \mathbb{R}^{n_{\boldsymbol{\theta}}} \to \mathbb{R}$ is a regularization term to enforce desirable properties on the weights.

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

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

2.2. Robust Optimization. Annie: todo

Adversarial training was introduced to promote robustness in DNNs. Instead of solving the optimization problem (2.1) at each data point exactly, a perturbation $\boldsymbol{\delta}_x$ is introduced to give the model additional information about the neighborhood around the point. The hope is that this additional information will allow the model to be less able to learn the bias in the original data. However, this perturbation introduces computational complexity- we now have a two layer optimization problem.

$$\min_{\boldsymbol{\theta}} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \left[\max_{\|\boldsymbol{\delta}_\mathbf{x}\|\leq r} L(f_{\boldsymbol{\theta}}(\mathbf{x}+\boldsymbol{\delta}_\mathbf{x}),\mathbf{y}) \right] + R(\boldsymbol{\theta}) \tag{2.2}$$

The idea is to perturb the inputs $\mathbf{x}$ by $\boldsymbol{\delta}_\mathbf{x}$ and maximize the size of the perturbation $\|\boldsymbol{\delta}_\mathbf{x}\|$ (inner optimization problem) while optimally fitting the data (outer minimization problem). In some sense, we build neighborhoods of radius r around our training points where we can similarly rely on the resulting classification. This is also called *robust optimization*.

The complexity of our new min-max problem is a large consideration for the applicability of our results to large scale real-world situations. If we do not solve the inner optimization efficiently, it may not be worth any potential improvements of fairness because of unrealistic computational times on real-world datasets. To address this, we investigate using second order information to avoid more computationally expensive first-order methods. There is also the question of how well we actually need to solve this inner optimization problem to see an improvement in fairness. We may be able to see similar results by skipping the relatively expensive task of solving the problem well and, for instance, simply guessing $\boldsymbol{\delta}_x$.

For each training sample $(\mathbf{x},\mathbf{y}) \in \mathcal{T}$, we must solve the inner constrained maximization problem of (2.2) well. This means we must satisfy the first-order optimality conditions, which appear as follows. We first setup the Lagrangian

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

where λ is a Lagrangian multiplier and we use an equivalent, differentiable version of the constraint. The perturbation $\boldsymbol{\delta}_\mathbf{x}$ that maxizimizes the inner optimization problem of (2.2) necessarily satisfies the Karush-Kuhn-Tucker conditions [6]

$$\nabla_{\boldsymbol{\delta}_\mathbf{x}} \mathcal{L}(\boldsymbol{\delta}_\mathbf{x},\lambda) = \nabla_{\boldsymbol{\delta}_\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}+\boldsymbol{\delta}_\mathbf{x}),\mathbf{y}) + \lambda\boldsymbol{\delta}_\mathbf{x} = \mathbf{0} \tag{2.4a}$$

$$\|\boldsymbol{\delta}_\mathbf{x}\|_2 \leq r \tag{2.4b}$$

$$\lambda \geq 0 \tag{2.4c}$$

$$\lambda(\|\boldsymbol{\delta}_\mathbf{x}\|_2 - r) = 0 \tag{2.4d}$$

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

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

$$+ \left[\nabla_{\boldsymbol{\theta}'}\boldsymbol{\delta}_\mathbf{x}(\boldsymbol{\theta}')\nabla_{\boldsymbol{\delta}_\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}+\boldsymbol{\delta}_\mathbf{x}(\boldsymbol{\theta}')),\mathbf{y})\right]_{\boldsymbol{\theta}'=\boldsymbol{\theta}}. \tag{2.6}$$

The first term in the expansion above is the traditional gradient that we want to preserve. The second term comes from considering the perturbation as a function of the network weights, $\boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta})$. If we solve the inner optimization problem well and thereby satisfy the KKT conditions, we can ignore the contribution of the second term. 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. From the primal feasibility condition (2.4b), we know that the perturbation must satisfy the constraint, even when undergoing changes. Specifically, the changes incurred from $[\nabla_{\boldsymbol{\theta}'} \boldsymbol{\delta}_{\mathbf{x}}(\boldsymbol{\theta}')]_{\boldsymbol{\theta}'=\boldsymbol{\theta}}$ cannot violate the constraint. Liz: Still working on this. Can derive for affine layers with logistic regression.

2.3. Fairness.

Allen: I got this!

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 the true labels and the classifier's predictions. In all of our experiments, the sensitive attribute is binary ($s = 0$ or $s = 1$), the true label is binary ($y = 0$ or $y = 1$), and the classifier prediction is binary ($\hat{y} = 0$ or $\hat{y} = 1$). For convenience, for 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 classification.

2.3.1. Independence.

For a classifier to satisfy independence simply means that the classifier's prediction, $\hat{Y}$, is uncorrelated with the sensitive attribute s. It requires an equal rate of positive classifications across all sensitive groups. For instance, if the sensitive attribute s were race (white or Black) and the value of $\hat{Y}$ were being used to determine whether or not to hire applicants to a job, satisfying independence would mean that if the classifier says to hire 20% of white applicants, then it also says to hire 20% of Black applicants. We can define independence as follows:

$$P(\hat{Y} = 1|s = 0) = P(\hat{Y} = 1|s = 1) = P(\hat{Y} = 1) \tag{2.7}$$

An interesting alternative interpretation of independence reverses the above conditional probabilities. The idea that $\hat{Y}$ is independent of s also can be expressed as

$$P(s = 1|\hat{Y} = 0) = P(s = 1|\hat{Y} = 1) = p(s = 1) \tag{2.8}$$

This can be interpreted as saying that the proportion of individuals of a certain classification who belong to a particular sensitive group should be equal to that sensitive group's proportion of the overall population being classified. For instance, if race is the sensitive attribute, independence would mean that if 15% of applicants in the applicant pool to a job are Black, then it is also true that 15% of hired ($\hat{Y} = 1$) applicants are Black.

2.3.2. Separation.

Separation is similar to independence, except that for separation to be satisfied means that $\hat{Y}$ is conditionally independent of s given the value of Y. In order for separation to be satisfied, the following two equalities have to hold:

$$P(\hat{Y} = 1|Y = 1, s = 0) = P(\hat{Y} = 1|Y = 1, s = 1) \tag{2.9}$$

$$P(\hat{Y} = 1|Y = 0, s = 0) = P(\hat{Y} = 1|Y = 0, s = 1) \tag{2.10}$$

If we think of the true label Y is representing an entity's qualifications, or whether or not an entity is "deserving" of being classified positively, separation can be thought of as a meritocratic fairness metric that says that if individuals who belong to different sensitive groups have the same qualifications, their probabilities of being classified positively should be equal. Using hiring and race as an example, separation would be satisfied if 90% of qualified white applicants are properly hired at the same time as 90% of qualified Black applicants are hired, and also 5% of unqualified white applicants were falsely hired at the same time as 5% of unqualified Black applicants were falsely hired. This sheds light on the fact that separation, instead of requiring equal rates of positive classification among sensitive groups like independence does, separation requires equal rates of true and false positive classification among sensitive groups. This can be seen because $P(\hat{Y} = 1|Y = 1)$ and $P(\hat{Y} = 1|Y = 0)$ represent the true and false positive rates, respectively, of the classifier. Unlike independence, which enforces that the classification be independent of the sensitive attribute completely, separation allows the classification $\hat{Y}$ to be correlated with s if the true label Y happens to be correlated with s.

2.3.3. Sufficiency. The equations for sufficiency are essentially the same as those for separation, but with Y and $\hat{Y}$ reversed:

$$P(Y = 1|\hat{Y} = 1, s = 0) = P(Y = 1|\hat{Y} = 1, s = 1) \tag{2.11}$$

$$P(Y = 1|\hat{Y} = 0, s = 0) = P(Y = 1|\hat{Y} = 0, s = 1) \tag{2.12}$$

Sufficiency can be interpreted as requiring that given two individuals in different sensitive groups with the same classification, their probabilities of having a given true label are the same. Essentially, what this means is that the output of a classifier is equally useful for predicting an individual's true label across sensitive groups. The proportion of positive classifications that truly correspond to positive labels, as well as the proportion of negative classifications that truly correspond to negative labels, is equal across sensitive groups. Going back to the idea of true labels representing qualifications, a real world example of sufficiency being satisfied would look like this: if 85% of white individuals who are hired for a job were indeed qualified, then 85% of hired Black individuals should be qualified as well; likewise, if 90% of white individuals who weren't hired were truly unqualified, the same proportion holds for Black individuals.

3. Our Approach.

3.1. Trust Region Subproblem. Our main algorithm focuses on solving a good approximation of the inner optimization problem of (2.2) well and efficiently using second-order information. For each training sample $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$,

we fix $\boldsymbol{\theta}$ and expand the loss function using a quadratic Taylor series expansion to get a quadratic approximation around $\mathbf{x}$ in the direction of $\boldsymbol{\delta}_x$.

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

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 $\boldsymbol{\delta}_x$ and λ.

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

We want to optimize $\boldsymbol{\delta}_x$. As a result of what is known in constrained optimization as the KKT conditions, we can deduce that the following first-order optimality conditions have to hold at an optimum for the problem [6]

1. $||\boldsymbol{\delta}_x|| \leq r$ (primal feasability)
2. $\lambda \geq 0$ (dual feasability)
3. $\nabla_{\boldsymbol{\delta}_x} \mathcal{L}(\boldsymbol{\delta}_x, \lambda) = 0$ (stationarity)
4. $\lambda(||\boldsymbol{\delta}_x|| - r) = 0$ (complementary slackness; at least one of conditions (1) and (2) holds with equality)

Applying condition (3) above, we can take the gradient of Equation 3.2 with respect to $\boldsymbol{\delta}_x$ and get that

$$\nabla_{\boldsymbol{x}} L(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) + \nabla^2_{\boldsymbol{x}} L(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) \boldsymbol{\delta}_{\boldsymbol{x}} + \lambda \boldsymbol{\delta}_{\boldsymbol{x}} = 0, \tag{3.3}$$

Rearranging the above equation, we get that the following relationship between $\boldsymbol{\delta}_{\boldsymbol{x}}$ and λ must hold:

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

3.1.1. Solving for λ. Using the complementary slackness condition (4), we can say furthermore that

$$\lambda(||\boldsymbol{\delta}_x(\lambda)|| - r) = 0 \text{ [6]} \tag{3.5}$$

This condition shows us that there are only two possibilities. The first is that $\lambda = 0$ and the solution to the constrained minimization problem is just the unconstrained minimum. In this case, we have $(\nabla^2_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}))\boldsymbol{\delta}_{\mathbf{x}} = -\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})$, which means that $\boldsymbol{\delta}_{\mathbf{x}}$ coincides with the familiar Newton step. If this unconstrained minimum satisfies condition (1), then we are all set - the unconstrained minimum is the solution to our constrained minimization problem. If, on the other hand, the Newton step does not satisfy condition (1), we must have $\lambda > 0$, in which case, condition (4) implies that $\|\boldsymbol{\delta}_{\mathbf{x}}\| = r$, meaning that our solution must lie on the boundary of our constraint region. In order for $\|\boldsymbol{\delta}_x\| - r = 0$, we must find a λ such that Equation (3.4) yields $\|\boldsymbol{\delta}_x(\lambda)\| = r$.

Before we solve for λ, we can algebraically manipulate Equation (3.4) to give us an easier equation to evaluate. Let $f(\mathbf{x})$ be our loss function, we note that the Hessian $\nabla^2 f(\mathbf{x})$ is a symmetric matrix, so we can take its eigendecomposition $\nabla^2 f(\mathbf{x}) = Q \Lambda Q^{-1}$, where Λ is a diagonal matrix of eigenvalues of the Hessian and Q is a matrix whose columns are eigenvectors. Because the Hessian is symmetric, Q is actually just an orthogonal matrix, and we can rewrite our decomposition as $\nabla^2 f(\mathbf{x}) = Q \Lambda Q^T$. We now use this decomposition and

the fact that Q is orthogonal to rewrite $\boldsymbol{\delta}_x(\lambda)$:

$$\begin{aligned}
\boldsymbol{\delta}_x(\lambda) &= -(\nabla^2 f(\mathbf{x}) + \lambda \mathbf{I})^{-1} \nabla f(\mathbf{x}) \\
&= -(Q \Lambda Q^T + \lambda \mathbf{I})^{-1} \nabla f(\mathbf{x}) \\
&= -(Q \Lambda Q^T + \lambda Q Q^T)^{-1} \nabla f(\mathbf{x}) \\
&= -\left(Q(\Lambda + \lambda \mathbf{I}) Q^T\right)^{-1} \nabla f(\mathbf{x}) \\
&= -\left(Q^T\right)^{-1} (\Lambda + \lambda \mathbf{I})^{-1} Q^{-1} \nabla f(\mathbf{x}) \\
&= -Q (\Lambda + \lambda \mathbf{I})^{-1} Q^T \nabla f(\mathbf{x}) \\
&= Q \mathbf{s}(\lambda),
\end{aligned}$$

$$\text{where } \mathbf{s}(\lambda) = -(\Lambda + \lambda \mathbf{I})^{-1} Q^T \nabla f(\mathbf{x})$$

It turns out that, because Q is orthogonal and its norm is equal to the identity matrix, we have that $||\boldsymbol{\delta}_x(\lambda)|| = ||Q\mathbf{s}(\lambda)|| = ||\mathbf{s}(\lambda)||$. This means that, instead of solving the problem $||\boldsymbol{\delta}_x(\lambda)|| = r$, we can instead solve the problem $||\mathbf{s}(\lambda)|| = r$ using the **bisection method**. This is very nice, because once we find the eigendecomposition of the Hessian, the matrix Q^T is very easy to compute, and because Λ is diagonal, $\Lambda + \lambda \mathbf{I}$ is also diagonal, which makes $\Lambda + \lambda \mathbf{I}$ computationally very easy to invert once we have the eigenvalues of the Hessian, perhaps much easier to invert than the original matrix $(\nabla^2 f(\mathbf{x}) + \lambda \mathbf{I})$.

3.1.2. The Bisection Method Bracket. To find a value for λ such that $\mathbf{s}(\lambda)$ lies on the boundary of the constraint, we build a univariate function $g(\lambda) := \|\mathbf{s}(\lambda)\|_2 - r$ and find a root of this function. Using (3.1.1), we can write g in terms of the eigenvalues of the Hessian $\nabla^2_{\mathbf{x}} f(\mathbf{x})$ as follows:

$$g(\lambda) := || - (\Lambda + \lambda \mathbf{I})^{-1} Q^T \nabla f(\mathbf{x})|| - r \tag{3.6}$$

Note that if the Hessian is positive semidefinite, g is a continuous function because the eigenvalues of the Hessian are nonnegative and $\lambda > 0$ to satisfy primal duality. However, when solving the maximization problem, the function g has singularities at negated eigenvalues of the Hessian.

We use the **bisection method** to find this root because it only requires function evaluations of g and no derivative information. The only requirement for the bisection method is that we begin with a bracket $[\alpha_{\text{low}}, \alpha_{\text{high}}]$ such that $g(\alpha_{\text{low}})$ and $g(\alpha_{\text{high}})$ have different signs, indicating a root exists within this interval. As a starting point, we can choose $\alpha_{\text{low}} = 0$. At $\lambda = 0$, the expression for $g(\lambda)$ simplifies to $||\nabla f(\mathbf{x})||_2 - r$. We expect the gradient at the point $\mathbf{x}$ to be nonzero (otherwise, we'd already be at an optimum), thus making $\alpha_{\text{low}} = 0$ a suitable choice. We need to choose a value for α_{high} to be big enough to ensure that the sign changes. Use the eigendecomposition of the Hessian, we obtain the upper bound for the bracket such

that $\|\mathbf{s}(\alpha_{\text{high}})\| < r$ as follows:

$$\|\mathbf{s}(\alpha_{\text{high}})\|_2 = \|(\Lambda + \alpha_{\text{high}}\mathbf{I})^{-1}Q^T\nabla f(\mathbf{x})\|_2 \qquad \text{by definition of } \mathbf{s}(\alpha_{\text{high}}) \tag{3.7}$$

$$\leq \|(\Lambda + \alpha_{\text{high}}\mathbf{I})^{-1}\|_2\|Q^T\nabla f(\mathbf{x})\|_2 \qquad \text{by submultiplicitivity of matrix norms} \tag{3.8}$$

$$\text{by 2-norm invariance of } Q \tag{3.9}$$

(3.10)

Note that this is an upper bound for $\|\mathbf{s}(\alpha_{\text{high}})\|_2$. Hence, we can set this upper bound equal to the radius and solve for α_{high} as follows:

$$\|\mathbf{s}(\alpha_{\text{high}})\| \leq \frac{1}{\lambda_{\min} + \alpha_{\text{high}}}\|\nabla f(\mathbf{x})\|_2 = r \qquad \Longrightarrow \qquad \alpha_{\text{high}} = \frac{\|\nabla f(\mathbf{x})\|_2}{r} + |\lambda_{\min}|. \tag{3.11}$$

Note that that the bisection method bracket does not require a tight upper bound. For this reason, we choose to increase α_{high} to ensure we have a good bracket by using

$$\alpha_{\text{high}} = \frac{n\|\nabla f(\mathbf{x})\|_2}{r} + |\lambda_{\min}|. \tag{3.12}$$

Liz: Not a good reason - I'll come up with something better.

With these values of α_{low} and α_{high}, we apply the **bisection method**. At each step, we calculate the midpoint $c = (\alpha_{\text{low}} + \alpha_{\text{high}})/2$ and evaluate the function g at this point. Depending on the sign of $g(c)$, we adjust our interval to be either [α_{low}, c] or [c, α_{high}] for the next iteration. We repeat this until the size of the interval is less than a predetermined tolerance level, and take the final midpoint as our approximation for the root.

3.2. Algorithm Analysis. Annie: todo

When solving the inner optimization problem using a second order approximation, we would like to know how well this approximation actually solves this problem.

3.2.1. Synthetic example.

For the synthetic example in section 4.2, an affine model $\mathbf{w}^\top\mathbf{x} + b$ with weight vector $\mathbf{w}$ and a scalar bias b, a logistic regression loss function L, and a sigmoid activation function σ is used.

$$L(\mathbf{x}, y) = -y\ln[\sigma(\mathbf{w}^\top\mathbf{x} + b)] - (1-y)\ln[1 - \sigma(\mathbf{w}^\top\mathbf{x} + b)], \quad \sigma(z) = \frac{1}{1+e^{-z}} \tag{3.13}$$

Introducing the perturbation $\boldsymbol{\delta}_x$ results in the following specific case of the inner optimization problem from section 2.2 (Annie: add ref to robust optimization eq here).

$$\max_{\|\boldsymbol{\delta}_x\| \leq r} \left[-y\ln(\sigma(\mathbf{w}^\top(\mathbf{x} + \boldsymbol{\delta}_x) + b)) - (1-y)\ln(1 - \sigma(\mathbf{w}^\top(\mathbf{x} + \boldsymbol{\delta}_x) + b))\right] \tag{3.14}$$

To solve this optimization problem without approximation, introduce a Lagrange multiplier λ with the constraint $\|\boldsymbol{\delta}_x\|^2 - r^2 \leq 0$.

$$\mathcal{L}(\boldsymbol{\delta}_x, \lambda) = -y\ln(\sigma(\mathbf{w}^\top(\mathbf{x}+\boldsymbol{\delta}_x)+b)) - (1-y)\ln(1-\sigma(\mathbf{w}^\top(\mathbf{x}+\boldsymbol{\delta}_x)+b)) + \frac{1}{2}\lambda(\|\boldsymbol{\delta}_x\|^2 - r^2) \tag{3.15}$$

Now, take the gradient with respect to $\boldsymbol{\delta}_x$. For simplicity, let $\sigma(\cdots) = \sigma(\mathbf{w}^\top(\mathbf{x}+\boldsymbol{\delta}_x)+b)$.

$$\begin{aligned}\nabla_{\boldsymbol{\delta}_x}\mathcal{L}(\boldsymbol{\delta}_x,\lambda) &= -y\mathbf{w}\frac{1}{\sigma(\cdots)}\sigma'(\cdots) - (1-y)(-\mathbf{w})\frac{1}{1-\sigma(\cdots)}\sigma'(\cdots) + \lambda\boldsymbol{\delta}_x \\ &= -y\mathbf{w}[1-\sigma(\cdots)] + (1-y)\mathbf{w}\sigma(\cdots) + \lambda\boldsymbol{\delta}_x \\ &= -y\mathbf{w} + \mathbf{w}\sigma(\cdots) + \lambda\boldsymbol{\delta}_x\end{aligned} \tag{3.16}$$

Setting (3.16) equal to zero, we obtain equation (3.17), which can be solved for optimal $\boldsymbol{\delta}_x$ using first order optimization conditions.

$$0 = -\mathbf{w}y + \mathbf{w}\sigma(\mathbf{w}^\top(\mathbf{x}+\boldsymbol{\delta}_x)+b) + \lambda\boldsymbol{\delta}_x \tag{3.17}$$

To compare the exact solution from equation (3.17) to the approximation made when solving using the trust region method of section 3.1, now find the second order approximation of the loss function $L(\mathbf{x}, y)$ by a Taylor expansion in $\mathbf{x}$ in the direction of $\boldsymbol{\delta}_x$.

$$L(\mathbf{x}+\boldsymbol{\delta}_x, y) \approx L(\mathbf{x},y) + \nabla_{\mathbf{x}}^\top L(\mathbf{x},y)\boldsymbol{\delta}_x + \frac{1}{2}\boldsymbol{\delta}_x^\top\nabla_{\mathbf{x}}^2 L(\mathbf{x},y)\boldsymbol{\delta}_x + \ldots \tag{3.18}$$

Computing the gradient and Hessian:

$$\begin{aligned}\nabla_{\mathbf{x}}L(\mathbf{x},y) &= -\mathbf{w}y + \mathbf{w}\sigma(\mathbf{w}^\top\mathbf{x}+b) \\ \nabla_{\mathbf{x}}^2 L(\mathbf{x})y) &= \mathbf{w}\sigma'(\mathbf{w}^\top\mathbf{x}+b)\mathbf{w}^\top\end{aligned} \tag{3.19}$$

To solve the optimization problem, we again introduce a Lagrange multiplier λ again subject to the constraint $\|\boldsymbol{\delta}_x\|^2 - r^2 \leq 0$.

$$\mathcal{L}(\boldsymbol{\delta}_x,\lambda) = L(\mathbf{x},y) + \nabla_{\mathbf{x}}^\top L(\mathbf{x},y)\boldsymbol{\delta}_x + \frac{1}{2}\boldsymbol{\delta}_x^\top\nabla_{\mathbf{x}}^2 L(\mathbf{x},y)\boldsymbol{\delta}_x + \frac{1}{2}\lambda(\|\boldsymbol{\delta}_x\|^2 - r^2) \tag{3.20}$$

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

$$\begin{aligned}\nabla_{\boldsymbol{\delta}_x}\mathcal{L}(\boldsymbol{\delta}_x,\lambda) &= \nabla_{\mathbf{x}}L(\mathbf{x},y) + \nabla_{\mathbf{x}}^2 L(\mathbf{x},y)\boldsymbol{\delta}_x + \lambda\boldsymbol{\delta}_x \\ 0 &= -\mathbf{w}y + \mathbf{w}\sigma(\mathbf{w}^\top\mathbf{x}+b) + \mathbf{w}\sigma'(\mathbf{w}^\top\mathbf{x}+b)\mathbf{w}^\top\boldsymbol{\delta}_x + \lambda\boldsymbol{\delta}_x\end{aligned} \tag{3.21}$$

Comparing equations 3.17 (LHS of 3.22) and 3.21 (RHS of 3.22), clearly the exact solution and the approximation are producing two different problems:

$$-\mathbf{w}y+\mathbf{w}\sigma(\mathbf{w}^\top(\mathbf{x}+\boldsymbol{\delta}_x)+b)+\lambda\boldsymbol{\delta}_x \neq -\mathbf{w}y+\mathbf{w}\sigma(\mathbf{w}^\top\mathbf{x}+b)+\mathbf{w}\sigma'(\mathbf{w}^\top\mathbf{x}+b)\mathbf{w}^\top\boldsymbol{\delta}_x+\lambda\boldsymbol{\delta}_x \tag{3.22}$$

How big of a difference is this? Getting rid of the common terms, here the terms causing the discrepancy:

$$\sigma(\mathbf{w}^\top(\mathbf{x}+\boldsymbol{\delta}_x)+b) \neq \sigma(\mathbf{w}^\top\mathbf{x}+b) + \sigma'(\mathbf{w}^\top\mathbf{x}+b)\mathbf{w}^\top\boldsymbol{\delta}_x \tag{3.23}$$

Now, apply a Taylor expansion centered at $\mathbf{x}$ in the ball $\mathbf{x}+\boldsymbol{\delta}_x$ to the LHS. For simplicity, let $\tilde{\mathbf{x}} = \mathbf{x}+\boldsymbol{\delta}_x$ and let $\mathbf{a} = \mathbf{x}$.

$$\sigma(\mathbf{w}^\top\tilde{\mathbf{x}}+b) \approx \sigma(\mathbf{w}^\top\mathbf{a}+b)+\nabla^\top\sigma(\mathbf{w}^\top\tilde{\mathbf{x}}+b)\Big|_{\tilde{\mathbf{x}}=\mathbf{a}}(\tilde{\mathbf{x}}-\mathbf{a})+\frac{1}{2}(\tilde{\mathbf{x}}-\mathbf{a})^\top\nabla^2\sigma(\mathbf{w}^\top\tilde{\mathbf{x}}+b)\Big|_{\tilde{\mathbf{x}}=\mathbf{a}}\boldsymbol{\delta}_x+\ldots \tag{3.24}$$

$$\text{where } \nabla^\top \sigma(\mathbf{w}^\top \tilde{\mathbf{x}} + b)\Big|_{\tilde{\mathbf{x}}=\mathbf{a}} = \Big[\mathbf{w}\sigma'(\mathbf{w}^\top \mathbf{a} + b)\Big]^\top = \sigma'(\mathbf{w}^\top \mathbf{a} + b)\mathbf{w}^\top \tag{3.25}$$
$$\text{and } \nabla^2 \sigma(\mathbf{w}^\top \tilde{\mathbf{x}} + b)\Big|_{\tilde{\mathbf{x}}=\mathbf{a}} = \mathbf{w}\sigma''(\mathbf{w}^\top \mathbf{a} + b)\mathbf{w}^\top$$

Now we obtain the following:

(3.26)
$$\sigma(\mathbf{w}^\top \tilde{\mathbf{x}} + b) \approx \sigma(\mathbf{w}^\top \mathbf{a} + b) + \sigma'(\mathbf{w}^\top \mathbf{a} + b)\mathbf{w}^\top(\tilde{\mathbf{x}} - \mathbf{a}) + \frac{1}{2}(\tilde{\mathbf{x}} - \mathbf{a})^\top \mathbf{w}\sigma''(\mathbf{w}^\top \mathbf{a} + b)\mathbf{w}^\top(\tilde{\mathbf{x}} - \mathbf{a}) + \dots$$

Converting back from $\mathbf{a} = \mathbf{x}$ and $\tilde{\mathbf{x}} - \mathbf{a} = \boldsymbol{\delta}_x$:

(3.27)
$$\sigma(\mathbf{w}^\top(\mathbf{x} + \boldsymbol{\delta}_x) + b) \approx \sigma(\mathbf{w}^\top \mathbf{x} + b) + \sigma'(\mathbf{w}^\top \mathbf{x} + b)\mathbf{w}^\top \boldsymbol{\delta}_x + \frac{1}{2}\boldsymbol{\delta}_x^\top \mathbf{w}\sigma''(\mathbf{w}^\top \mathbf{x} + b)\mathbf{w}^\top \boldsymbol{\delta}_x + \dots$$

We have recovered the RHS of (3.23)! The difference between the two is confined to the second-order and higher terms of (3.26).

For a function $f : \mathbb{R}^n \to \mathbb{R}$ that is $k+1$-times differentiable in a closed ball $B = \{\mathbf{y} \in \mathbb{R}^n : \|\boldsymbol{\delta}_x - \mathbf{y}\| \le a\}$ for some $a > 0$ the exact remainder of the Taylor series in this neighborhood is given by: (Annie: check citation, used wikipedia page)

(3.28)
$$f(\mathbf{x}) = \sum_{|\alpha| \le k} \frac{D^\alpha f(\mathbf{a})}{\alpha!}(\mathbf{x} - \mathbf{a})^\alpha + \sum_{|\beta| = k+1} R_\beta(\mathbf{x})(\mathbf{x} - \mathbf{a})^\beta$$

(3.29)
$$\text{where } R_\beta(\mathbf{x}) = \frac{|\beta|}{\beta!}\int_0^1 (1-t)^{|\beta|-1} D^\beta f(\mathbf{a} + t(\mathbf{x} - \mathbf{a}))dt, \quad \text{and } \alpha, \beta \text{ are multi-indices}$$

In our specific case, where $f(\mathbf{x}) = \sigma(\mathbf{w}^\top \tilde{\mathbf{x}} + b)$ and $k = 1$ (again with $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\delta}_x$ and $a = \mathbf{x}$), we have:

(3.30)
$$\sigma(\mathbf{w}^\top \tilde{\mathbf{x}} + b) = \underbrace{\sum_{|\alpha| \le 1} \frac{D^\alpha \sigma(\mathbf{w}^\top \tilde{\mathbf{x}} + b)\big|_{\tilde{\mathbf{x}}=\mathbf{a}}}{\alpha!}(\tilde{\mathbf{x}} - \mathbf{a})^\alpha}_{A} + \underbrace{\sum_{|\beta| = 2} R_\beta(\tilde{\mathbf{x}})(\tilde{\mathbf{x}} - \mathbf{a})^\beta}_{B}$$

Sum A in (3.30) is simply multi-index notation for the first two terms of the expansion in (3.26):

(3.31)
$$A = \sum_{|\alpha|=0} \frac{D^\alpha \sigma(\mathbf{w}^\top \tilde{\mathbf{x}} + b)\big|_{\tilde{\mathbf{x}}=\mathbf{a}}}{\alpha!}(\tilde{\mathbf{x}} - \mathbf{a})^\alpha + \sum_{|\alpha|=1} \frac{D^\alpha \sigma(\mathbf{w}^\top \tilde{\mathbf{x}} + b)\big|_{\tilde{\mathbf{x}}=\mathbf{a}}}{\alpha!}(\tilde{\mathbf{x}} - \mathbf{a})^\alpha$$

Let n be the dimension of the multi-indices α and β, which correspond to $\mathbf{x}$ and $\boldsymbol{\delta}_x$ being n-dimensional vectors with components $x_1, \dots, x_n$ and $\delta_{x,1}, \dots, \delta_{x,n}$ respectively, and let $\hat{\mathbf{e}}_1, \dots, \hat{\mathbf{e}}_n$ be linearly independent unit vectors in n-space, such that each component of a given $\hat{\mathbf{e}}_i$ is 0 except for the i-th.

The factorial of a multi-index is the product of the factorials of each component (so for α with components $\alpha_1, \dots, \alpha_n$, $\alpha! = \alpha_1! \cdot \alpha_2! \cdot \dots \cdot \alpha_n!$). For the cases in (3.31) where $\alpha = 0$ or 1, either all components $\alpha_1, \dots, \alpha_n = 0$, or all components are zero except one component $\alpha_i = 1$. Since 0! and 1! both equal 1, the product of any combination of these will always be 1, so $\alpha! = 1\ \forall \alpha = 0, 1$. Annie: Not sure if this paragraph is necessary:

$$
\begin{aligned}
A &= D^0\sigma(\mathbf{w}^\top\tilde{\mathbf{x}}+b)\Big|_{\tilde{\mathbf{x}}=\mathbf{a}}(\tilde{\mathbf{x}}-\mathbf{a})^0 + \sum_{i=1}^{n}\hat{\mathbf{e}}_i\cdot\nabla\sigma(\mathbf{w}^\top\tilde{\mathbf{x}}+b)\Big|_{\tilde{\mathbf{x}}=\mathbf{a}}(\tilde{x}_i-a_i) \\
&= \sigma(\mathbf{w}^\top\mathbf{a}+b) + \nabla^\top\sigma(\mathbf{w}^\top\tilde{\mathbf{x}}+b)\Big|_{\tilde{\mathbf{x}}-\mathbf{a}}(\tilde{\mathbf{x}}-\mathbf{a}) \\
&= \sigma(\mathbf{w}^\top\mathbf{a}+b) + \sigma'(\mathbf{w}^\top\mathbf{a}+b)\mathbf{w}^\top(\tilde{\mathbf{x}}-\mathbf{a})
\end{aligned} \tag{3.32}
$$

Now we must calculate sum B in (3.30) to find the remainder.

$$
\sum_{|\beta|=2} R_\beta(\tilde{\mathbf{x}})(\tilde{\mathbf{x}}-\mathbf{a})^\beta = \sum_{|\beta|=2}\left[\frac{2}{\beta!}\int_0^1 (1-t)D^\beta\sigma(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)dt\right](\tilde{\mathbf{x}}-\mathbf{a})^\beta \tag{3.33}
$$

Let i, j and k be indices from $1,\ldots,n$. There are two cases of possible β:

- **Case 1:** $\beta_i,\beta_j = 1, i\neq j$ and $\beta_k = 0$ for all k s.t. $k\neq i\neq j$, denoted $\beta^{(1,1)}$
- **Case 2:** $\beta_i = 2$ and $\beta_k = 0$ for all k s.t. $k\neq i$, denoted $\beta^{(2)}$

We can now write sum b as a combination of the two cases.

$$
\sum_{|\beta|=2} R_\beta(\tilde{\mathbf{x}})(\tilde{\mathbf{x}}-\mathbf{a})^\beta = \binom{n}{2}\sum_{|\beta^{(1,1)}|=2} R_\beta(\tilde{\mathbf{x}})(\tilde{\mathbf{x}}-\mathbf{a})^\beta + n\sum_{|\beta^{(2)}|=2} R_\beta(\tilde{\mathbf{x}})(\tilde{\mathbf{x}}-\mathbf{a})^\beta \tag{3.34}
$$

Examining the term $D^\beta\sigma(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)$:

$$
\begin{aligned}
D^\beta\sigma(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b) &= \partial^{\hat{\mathbf{e}}_i}\partial^{\hat{\mathbf{e}}_j}\sigma(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b) \\
&= \partial^{\hat{\mathbf{e}}_i}\left[\sigma'(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)\partial^{\hat{\mathbf{e}}_j}(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)\right] \\
&= \partial^{\hat{\mathbf{e}}_i}\left[\sigma'(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)\left(\hat{\mathbf{e}}_j\cdot\nabla(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)\right)\right] \\
&= \partial^{\hat{\mathbf{e}}_i}\left[\sigma'(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)(\hat{\mathbf{e}}_j\cdot t\mathbf{w})\right] \\
&= t(\hat{\mathbf{e}}_j\cdot\mathbf{w})\partial^{\hat{\mathbf{e}}_i}\left[\sigma'(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)\right] \\
&= t(\hat{\mathbf{e}}_j\cdot\mathbf{w})\sigma''(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)(\hat{\mathbf{e}}_i\cdot t\mathbf{w}) \\
&= t^2(\hat{\mathbf{e}}_j\cdot\mathbf{w})(\hat{\mathbf{e}}_i\cdot\mathbf{w})\sigma''(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b) \\
&= t^2 w_i w_j\sigma''(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b) \\
\text{and for } i=j:\quad &= t^2 w_i^2\sigma''(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b) \\
\text{which for both cases is just}\quad &= \mathbf{w}^\beta t^2\sigma''(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)
\end{aligned} \tag{3.35}
$$

- **Case 1:**

$$
\begin{aligned}
&\sum_{|\beta^{(1,1)}|=2}\left[\frac{2}{1!\cdot 1!\cdot 0!\cdot\ldots}\int_0^1 (1-t)D^\beta\sigma(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)dt\right](\tilde{\mathbf{x}}-\mathbf{a})^\beta \\
&= \sum_{|\beta^{(1,1)}|=2}\left[2\mathbf{w}^\beta\int_0^1 (1-t)t^2\sigma''(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)dt\right](\tilde{\mathbf{x}}-\mathbf{a})^\beta
\end{aligned} \tag{3.36}
$$

- **Case 2:**

$$
\begin{aligned}
&\sum_{|\beta^{(2)}|=2}\left[\frac{2}{2!\cdot 0!\cdot\ldots}\int_0^1 (1-t)D^\beta\sigma(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)dt\right](\tilde{\mathbf{x}}-\mathbf{a})^\beta \\
&= \sum_{|\beta^{(2)}|=2}\left[\mathbf{w}^\beta\int_0^1 (1-t)t^2\sigma''(\mathbf{w}^\top(\mathbf{a}+t(\tilde{\mathbf{x}}-\mathbf{a}))+b)dt\right](\tilde{\mathbf{x}}-\mathbf{a})^\beta
\end{aligned} \tag{3.37}
$$

It turns out that the integral in each case is the same. To evaluate it, let $p = \mathbf{w}^\top(\tilde{\mathbf{x}} - \mathbf{a})$ and $s = \mathbf{w}^\top \mathbf{a} + b$

$$\int_0^1 (1-t)t^2\sigma''(pt+s)dt = \int_0^1 t^2\sigma''(pt+s)dt + \int_0^1 t^3\sigma''(\mathbf{w}^\top(pt+s)dt \tag{3.38}$$

$$\int_0^1 t^2\sigma''(pt+s)dt = \left|_0^1 t^2\frac{1}{p}\sigma'(pt+s) - 2t\frac{1}{p^2}\sigma(pt+s) = \frac{1}{p}\sigma'(p+s) - 2\frac{1}{p^2}\sigma(p+s)\right. \tag{3.39}$$

$$\begin{aligned}\int_0^1 t^3\sigma''(pt+s)dt &= \left|_0^1 t^3\frac{1}{p}\sigma'(pt+s) - 3t^2\frac{1}{p^2}\sigma(pt+s) + 6t\frac{1}{p^3}\ln(e^{pt+s}+1)\right. \\ &= \frac{1}{p}\sigma'(p+s) - 3\frac{1}{p^2}\sigma(p+s) + 6\frac{1}{p^3}\ln(e^{p+s}+1)\end{aligned} \tag{3.40}$$

$$\Rightarrow \int_0^1 (1-t)t^2\sigma''(pt+s)dt = 2\frac{1}{p}\sigma'(p+s) - 5\frac{1}{p^2}\sigma(p+s) + 6\frac{1}{p^3}\ln(e^{p+s}+1) \tag{3.41}$$

Since p and s are constants, the entire integral does not vary with β so it can be factored out of the sum. Let I represent the value of the integral.

$$\begin{aligned}\sum_{|\beta|=2} R_\beta(\tilde{\mathbf{x}})(\tilde{\mathbf{x}} - \mathbf{a})^\beta &= \binom{n}{2}\sum_{|\beta^{(1,1)}|=2} 2\mathbf{w}^\beta I(\tilde{\mathbf{x}} - \mathbf{a})^\beta + n\sum_{|\beta^{(2)}|=2}\mathbf{w}^\beta I(\tilde{\mathbf{x}} - \mathbf{a})^\beta \\ &= 2I\frac{n(n-1)}{2}\sum_{|\beta^{(1,1)}|=2}\mathbf{w}^\beta(\tilde{\mathbf{x}} - \mathbf{a})^\beta + In\sum_{|\beta^{(2)}|=2}\mathbf{w}^\beta(\tilde{\mathbf{x}} - \mathbf{a})^\beta \\ &= In^2\sum_{|\beta|=2}\mathbf{w}^\beta(\tilde{\mathbf{x}} - \mathbf{a})^\beta\end{aligned} \tag{3.42}$$

Annie: leaving off here for now, edits to come

Resubstituting $p = \mathbf{w}^\top(\tilde{\mathbf{x}} - \mathbf{a})$ and $p + s = \mathbf{w}^\top\tilde{\mathbf{x}} + b$ the integral becomes:

$$2\frac{1}{\mathbf{w}^\top(\tilde{\mathbf{x}} - \mathbf{a})}\sigma'(\mathbf{w}^\top\tilde{\mathbf{x}}+b) - 5\frac{1}{(\mathbf{w}^\top(\tilde{\mathbf{x}} - \mathbf{a}))^2}\sigma(\mathbf{w}^\top\tilde{\mathbf{x}}+b) + 6\frac{1}{(\mathbf{w}^\top(\tilde{\mathbf{x}} - \mathbf{a}))^3}\ln(e^{\mathbf{w}^\top\tilde{\mathbf{x}}+b}+1) \tag{3.43}$$

3.3. Other Methods. We also implemented two methods known as Random Perturbation and Projected Gradient Descent (PGD) (a) to examine whether or not it is important to solve the inner optimization problem well and (b) to verify that our second-order trust region subproblem approach has greater computational efficiency than using purely first-order information.

3.3.1. Random Perturbation. We wanted to verify that solving the inner optimization problem well with optimal perturbations - and not just random ones - is important for seeing fairness improvement with robust training. To implement a random perturbation, we added to our code a solver for $\boldsymbol{\delta}_x$ in our inner optimization problem that simply, for each data point, generated each component of $\boldsymbol{\delta}_x$ as a random number from a normal distribution and then scaled this random vector to have a norm equal to the perturbation radius. While a manual seed is set for our whole repository, we do not set a manual seed in the Random Perturbation solver so that different perturbations are chosen at different data points.

3.3.2. Projected Gradient Descent (PGD). In order to verify that our second-order trust region subproblem approach is computationally faster than using purely first-order information, we also implemented a version of gradient descent for our inner optimization problem. Because this inner problem has the constraint $||\boldsymbol{\delta}_x|| \leq r$, we could not use vanilla gradient descent - we need to ensure that we satisfy the constraint as we are optimizing our objective function value. We thus used Projected Gradient Descent (PGD), where we take a step in the direction we would take with gradient descent if we were solving an unconstrained optimization problem and then project the point we reach onto the constraint set, using this projected point as our next iterate. Mathematically, this looks like:

$$\boldsymbol{\delta}_{\boldsymbol{x}}^{(k+1)} = P\left[\boldsymbol{\delta}_{\boldsymbol{x}}^{(k)} + \alpha^{(k)} \cdot \nabla_{\boldsymbol{x}} L(f_{\boldsymbol{\theta}}(\boldsymbol{x} + \boldsymbol{\delta}_{\boldsymbol{x}}), \boldsymbol{y})\right] \tag{3.44}$$

where $\boldsymbol{\delta}_{\boldsymbol{x}}$ is the kth iterate, $\alpha^{(k)}$ is the step size at the kth iteration, and P is the projection operator $P(x) = \text{argmin}_{z:||z||_2 \leq r} ||x - z||_2^2$. As one can show easily using multivariable calculus or Lagrange multipliers, this projection operator turns out to be very simple, just returning the point inputted into it if the point is already satisfies the ball constraint, and scaling the point inward to the boundary of the ball if the starting point is outside the ball. In particular,

$$P(x) = x \min\{1, \frac{r}{||x||_2}\}. \tag{3.45}$$

With this projection operation specified, all that needs to be discussed is how we evaluated the gradient, chose a step size, and chose when to stop. To evaluate the gradient efficiently, we used the HessQuik package provided to us. As for a step size, in any given iteration, we started with a given fixed step size, and if this step size did not achieve a more optimal objective function value, we decreased the step size by a factor of 1.5 repeatedly until getting to a step size where our step would achieve a more optimal objective function value. We set up our PGD to stop and declare our line search "broken" if we divided our step size by 1.5 enough times without finding a step size that achieved a more optimal objective function value. We called this maximum number of times we would reduce the step size on a given iteration as a parameter called max_divisions and set its default value equal to 40.

In unconstrained optimization, a natural stopping point for gradient descent is to go until the norm of the gradient is less than or equal to some tolerance, since the gradient must be 0 at an optimum. In this case, however, we could not use a stopping criteria like this, since we have a constrained optimization problem. Instead, we chose a stopping criteria based on the idea that if we are close to an optimum, our iterates should not be changing much. Thus, we chose a stopping criterion for our PGD of stopping whenever we got to a point where the norm of the difference between the current iterate and the next iterate that would be specified by our algorithm is less than or equal to a certain tolerance, which we chose to be equal to 10^{-4}. We also had PGD stop after a certain maximum number of iterations, which we set to 5000.

We set up a quadratic test example to verify that PGD gave us the same optimum as the trust region subproblem and did indeed find that the two approaches yielded the same solution. We also clearly saw later when we were doing our final experiments that the fairness metrics we got with PGD were identical to the results we were obtaining with the trust region subproblem, giving us confidence that we had implemented PGD correctly.

4. Numerical Results.

4.1. Setup.

Liz: Someone describe metrics!

4.2. Synthetic Data.

Allen: This is my wheelhouse! The primary dataset we used for carrying out numerical experiments was a synthetic dataset[3]. The context is that individuals of two different sensitive groups, namely A and 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. We initially set up the data in a fair way; see Figure 1 for details. However, we introduced unfair bias into the data by shifting all Bs to have higher scores and shifting all As in the opposite direction to have lower scores; see Figure 2 for details. In the real world, this could be a manifestation of some kind of phenomenon where Bs happen to belong to a wealthy socioeconomic class and can afford training that boosts their scores, whereas As don't have this opportunity.

4.2.1. Unfair2D Fairness Results.

We compared nonrobust training on this synthetic example with robust training (using the trust region subproblem 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 3, as well as plots of the training and testing accuracy with radius. (For clarity, in the plots, "conditioned on $Y = 0$ or $\hat{Y} = 0$ would, for separation, mean the difference $|P(\hat{Y} = 1|Y = 0, s = 1) - P(\hat{Y} = 1|Y = 0, s = 0)|$, and likewise for sufficiency with the roles of Y and $\hat{Y}$ reversed.) Allen: I know it may be said that this is something that should go in a caption, but there is a lot of whitespace anyway, it is not very long, and when I tried to put this in the caption with one of the six plots, two of the plots that should be next to each other came to have a vertical offset, which looked no good.

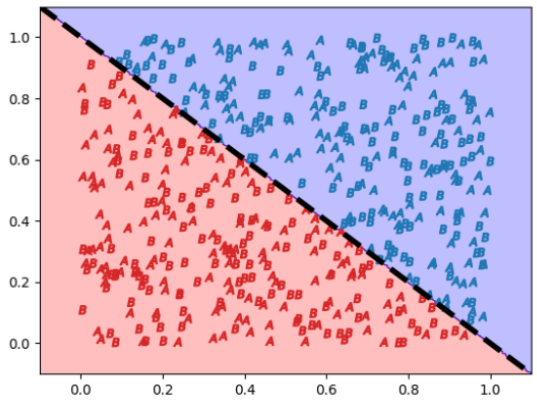


Figure 1: Original Data. For every data point, we generated each numeric score uniformly at random on [0, 1]. We made a classifier $x_2 = -x_1 + 1$ (shown as the dashed line) and gave a blue label ($Y = 1$) of to all individuals lying above this line and a red label ($Y = 0$) to all individuals below. Each point was randomly assigned to be A or B with 0.5 probability each. At this point, the dashed classifier is fair - it satisfies independence, since an individual's sensitive attribute is independent of what side of the classifier they lie on. Also, since it is error-free, the classifier satisfies separation with equal true positive rates of 1 and false positive rates of 0 for both groups, and it satisfies sufficiency, since for both groups, it is perfectly predictive of their true labels.

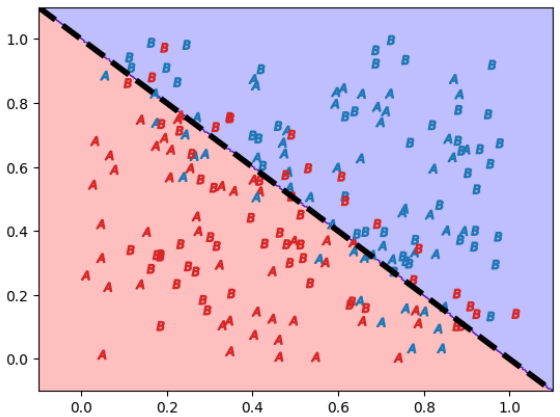
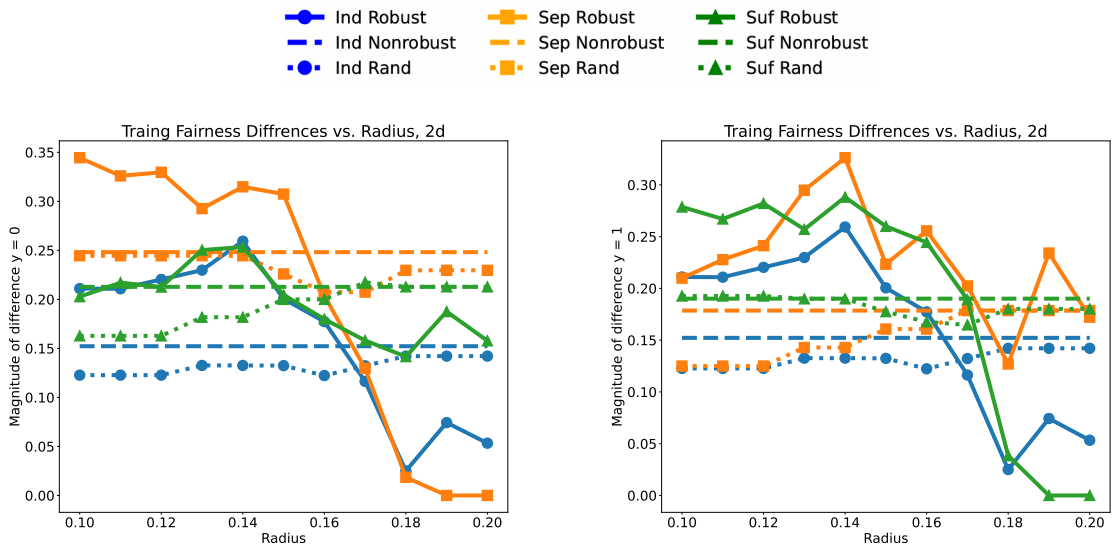
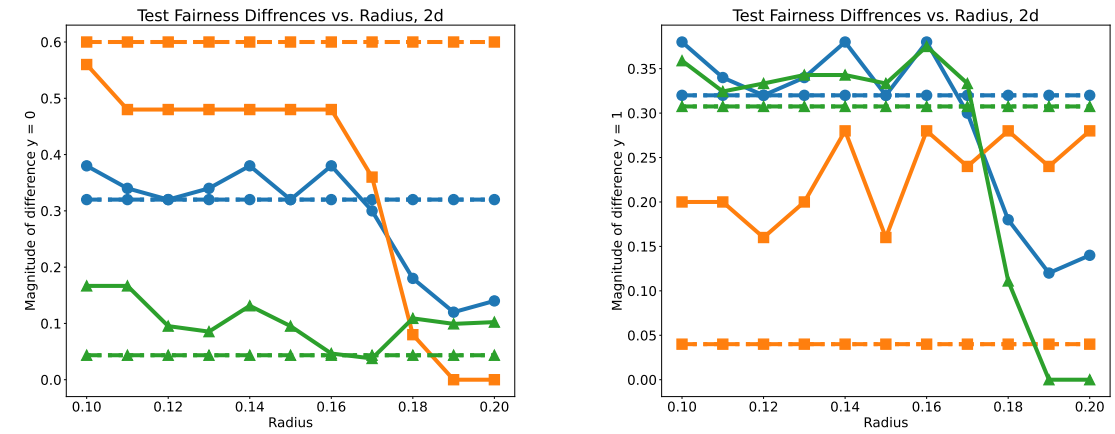


Figure 2: Unfair data after shift; the dashed classifier is now unfair. Because of the shift, sensitive attribute is no longer independent of what side of the classifier you are on, violating independence. Since all false positives (reds in blue region) are Bs and all false negatives (blues in red region) are As, we cannot have equality of false negative and positive rates, violating separation. On a related note, sufficiency is violated, because while all As in the blue region truly have a blue label, this is not true of Bs, and while all Bs in the red region truly have a red label, this is not true of As.



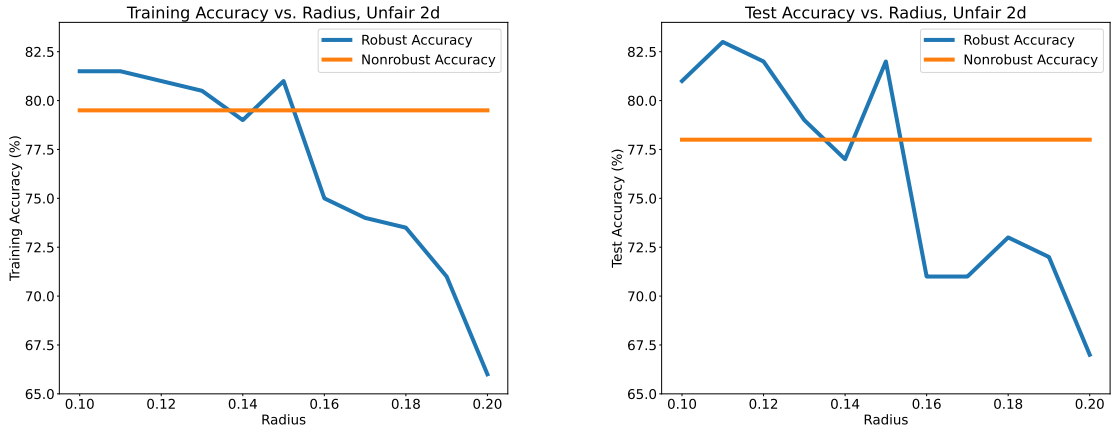
(a) Training differences conditioned on Y or $\hat{Y}$ equal to 0

(b) Training differences conditioned on Y or $\hat{Y}$ equal to 1



(c) Test differences conditioned on Y or $\hat{Y}$ equal to 0

(d) Test differences conditioned on Y or $\hat{Y}$ equal to 1



(e) Training accuracy vs. radius

(f) Test accuracy vs. radius

Figure 3: Fairness trends for synthetic example for varying radii

Looking at the plots, for many radii in the lower end of the 0.1 to 0.2 range, many of the fairness differences are worse in the training data with robust training than with nonrobust.

However, there is some promise for fairness improvement. In both Figure 3a and Figure 3b, the differences for all three fairness metrics show an overall downward trend, and in both Figure 3c and Figure 3d, the differences for 2 out of 3 fairness metrics exhibit a downward trend. This could be partly due to the fact that we, by default, had 50 instead of 200 data points in the test set. Nonetheless, this great quantity of cases where fairness differences show a downward trend - all of them for the training data and a majority for the test data - as the radius increases seem to show promise that, while robust training may worsen fairness for a radius too small, fairness improvement is possible if you increase the radius far enough. Strikingly, in many of the plots, it appears consistently to be a radius between 0.14 and 0.17 where fairness differences for robust training first get better than those for nonrobust training in cases where there is a downward trend. Interestingly, at a radius of 0.18, all of the robust fairness differences are better than the corresponding nonrobust ones in the training data, as can be seen in the tables below. Below is also a visualization of what the nonrobust trained classifier looks like versus a robust one with perturbation radius equal to 0.18.

Non-robust classifier

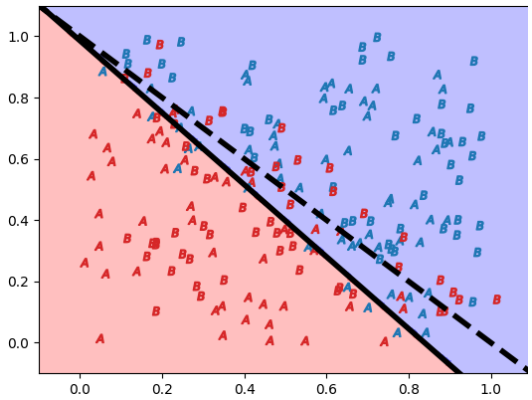


Figure 4: Nonrobust Classifier

Robust classifier (r=0.18)

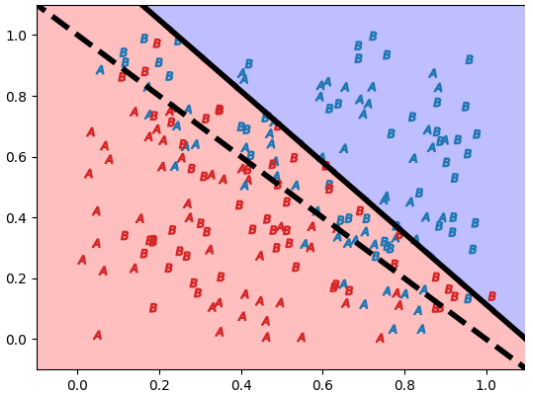


Figure 5: Robust classifier

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%

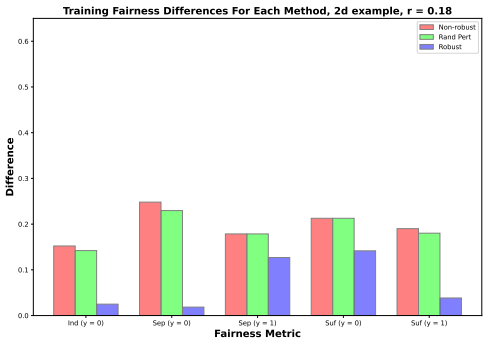
Diff:	Y=0 —S1-S0—	Y=1 —S1-S0—
Ind.	0.025	0.025
Sep.	0.019	0.127
Suff.	0.142	0.038

Training Accuracy: 73.5%
Test Accuracy: 73.0%

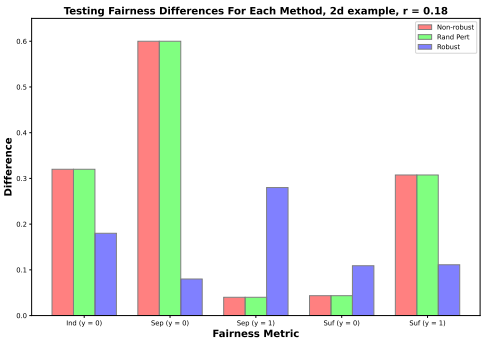
As the above information indicates, going up to a radius of 0.18 in this example, while improving fairness significantly, does lead to a decrease in accuracy from around 78% to 73%,

though not a huge decrease. There seems to be a fairness/accuracy tradeoff; as one can see in Figure 3e and Figure 3f, both the training and test accuracy show a downward trend as the radius increases.

Another comment worth making is that there is an advantage to solving the inner optimization problem well over just using random perturbation. Looking back at Figure 3, it is consistent throughout figures (a) - (d) that the fairness metric differences are either the same in random perturbation as for nonrobust training or "tend toward" the nonrobust training differences. By "tend toward" it is meant that if the differences for random perturbation start near the nonrobust differences, they stay near, or if they don't start near, they get closer and closer to the nonrobust differences as the radius increases. This stands in contrast to the robust training, where in many cases where fairness differences have a downward trend, the robust training differences are initially high and then get a lot lower. They do not show any behavior of settling toward the nonrobust differences. What this suggests is that if you select a radius where robust training shows a significant improvement in fairness, you do not see this by just by using random perturbations, because in these situations, the differences for random perturbation are generally relatively close to the nonrobust differences when compared to the robust differences. These observations are borne out in the following two bar graphs of fairness metric differences for a radius of 0.18:



(a) Differences for radius 0.18 for training data. Left bar is for nonrobust; middle for Rand Pert; and right for robust.



(b) Differences for radius 0.18 for test data. Left bar is for nonrobust; middle for Rand Pert; and right for robust.

Notice that, in the above graphs, the fairness differences for random perturbation (green) are consistently a lot closer to the corresponding nonrobust fairness differences (red) than the corresponding robust differences (blue) are. Clearly, in all of the cases in the above graphs where the blue bar (robust training) shows a marked improvement in a fairness metric relative to the corresponding red bar (nonrobust training), the green bar (random perturbation) only shows marginal fairness improvement, if at all, and is for all intents and purposes about the same as the red bar.

We can garner an interesting piece of insight about what robust training is doing from the classifiers shown in Figure 4 and Figure 5. Essentially, what is happening is that our robust optimization is improving fairness by raising the bar and only giving a positive classification to the most qualified individuals. It shifts the classification line outward, eliminating nearly all of the false positive Bs that would exist with the dashed classifier and greatly expanding the quantity of blue Bs in the red region, thereby equalizing false negative rates. In fact, as we saw, if you increased the radius even further to 0.2, the line is shifted further outward, to the point that if you do 20 epochs of training with a radius of 0.2, the robust classifier classifies no one positively. These observations are interesting, because one normally tends to think of improving fairness by giving a leg up to the disadvantaged group, in this case A. What we see here, however, is something completely different - the robust classifier does not help any As at all; the fairness improvement comes entirely from the robust classifier hurting the advantaged group, Bs. While this does not necessarily provide an indication of how robust training would work on every dataset, it does illustrate an example of how robust optimization may work to improve fairness in a surprising way that would not be considered socially desirable.

4.3. Adult Dataset. [7]Hung: I'll describe the data!

The Adult dataset is a popular dataset used in fairness-aware classification studies. It 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 males, white, and people in the 25-60 years old age group. The dataset contains 48,842 instances and each instance

is described using 15 attributes. These attributes are age, workclass, final weight (fnlwgt), education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country, and income. Of these, 6 are numerical, 7 are categorical, and 2 are binary attributes. We want to look at the 5 continuous numerical attribute (age, education-num, capital-gain, capital-loss, hours-per-week) for analysis and normalization. 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.

4.3.1. Adult Fairness Trends. Unlike our synthetic data, the adult example yielded mixed results in terms of fairness improvement. For the training data, Figure 7a and Figure 7b shows that only three out of the six differences were measured to be better with robust training. There was also a similar result for the testing data as seen in Figure 7c and Figure 7d. It is important to recognize that, in both the training and testing dataset, the robust independence differences at the starting radius of .10 were better than its non-robust counterpart so it is hard to say that robust training improved independence in either case. Additionally, if we look at the robust sufficiency trend of Figure 7c, we can see that robust differences remain smaller than the non-robust differences if it starts off smaller. Despite only having a 50% improvement rate, there were still two positive trends that remained the same: the majority of the fairness metrics exhibit a downward trend and, when there is an improvement, robust optimization out performs random perturbation.

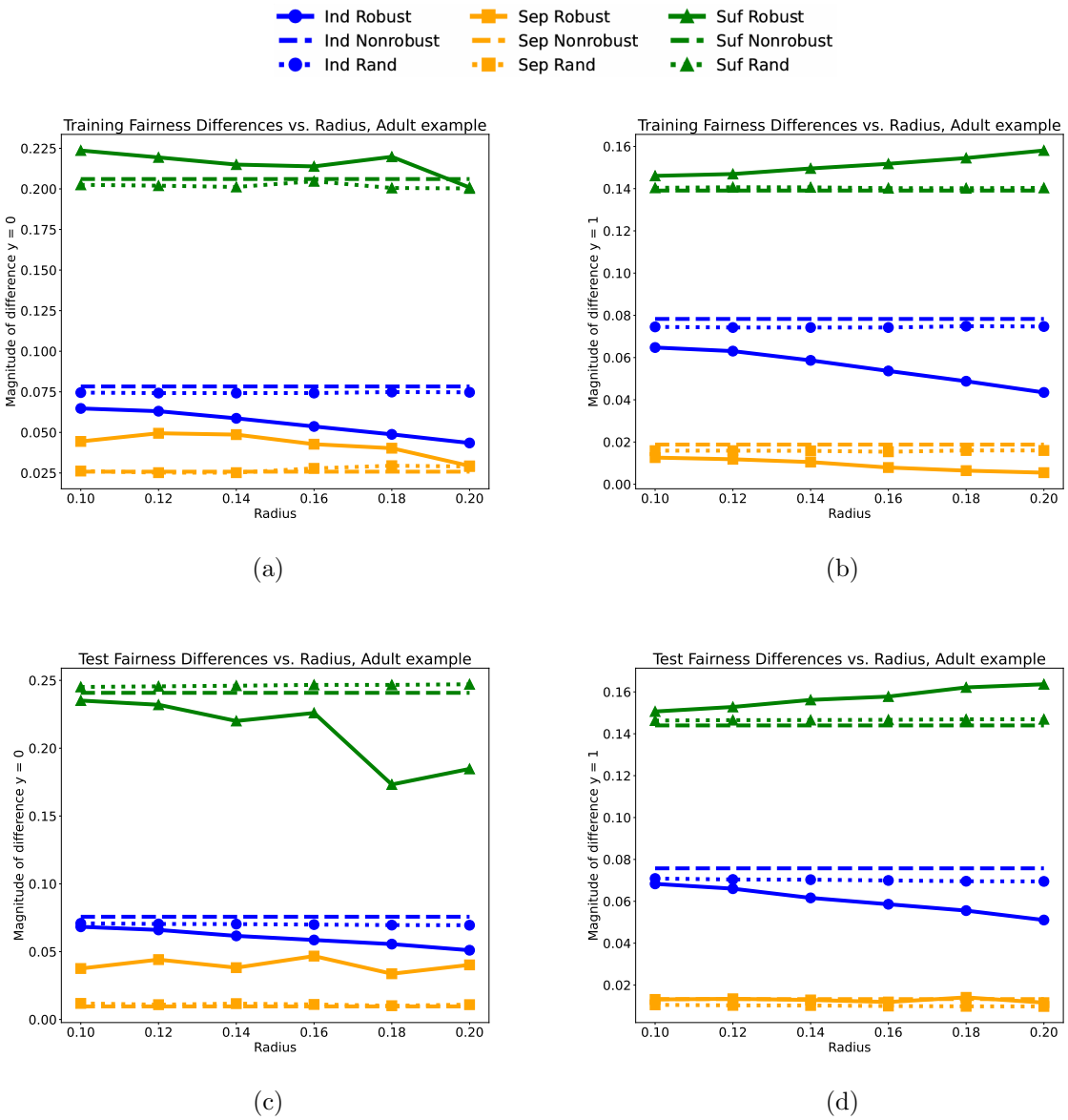
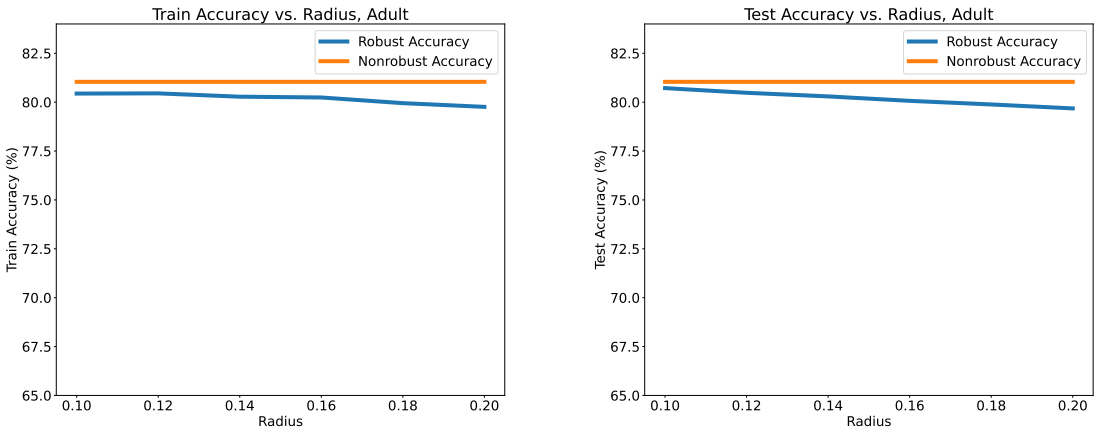


Figure 7: Fairness trends in the adult dataset for nonrobust and robust training with six different perturbation radii ranging from .1 to .2 with increments of .02. The experiment ran with 10 epochs and a learning rate of .01.

4.3.2. Adult Accuracy Trends.



The decrease in accuracy from non-robust to robust training is still apparent in this example. From the two plots above, both the training and test robust accuracy exhibit a downward trend as the radius increases. However, unlike the synthetic example, the decay appears to be linear and does not spike at certain radii. Furthermore, robust training on this dataset does not yield better accuracy for smaller radii like what was observed for our 2D example. Looking at the fairness trends along with the accuracy trends for the adult dataset, we can conclude that there is still a trade-off for improving some of the fairness metrics by lowering the accuracy of our classifier.

4.4. LSAT Data. Hung: Data is from: Link Question: How to cite?

Another extension to a real-world dataset comes from the Law School Admissions Council (LSAC). The dataset was collected to explore the reasons behind low bar passage rates among racial and ethnic minority examinees. For our purpose, we want to 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, and to analyze the fairness of the model's predictions. We are using grade point average (ugpa) and LSAT scores because they are the strongest predictors for passing the bar examination. We are interested in five features of the dataset:

- **dnn_bar_pass_prediction:** The prediction made by a Deep Neural Network (DNN) model about whether the student will pass the bar exam.
- **gender:** The gender of the student.
- **lsat:** The score obtained by the student on the LSAT.
- **pass_bar:** The truth indicating whether the student actually passed the bar.
- **race:** The race of the student.
- **ugpa:** The undergraduate grade point average of the student.

In our experiment, the race feature is binarized to indicate whether the student is white or not. This will act as our sensitive attribute. We also ensures that the training set has a balanced representation of all four groups defined by the two binary attributes: whether the student passed the bar exam and whether the student is white.

4.4.1. LSAT Fairness Trends.

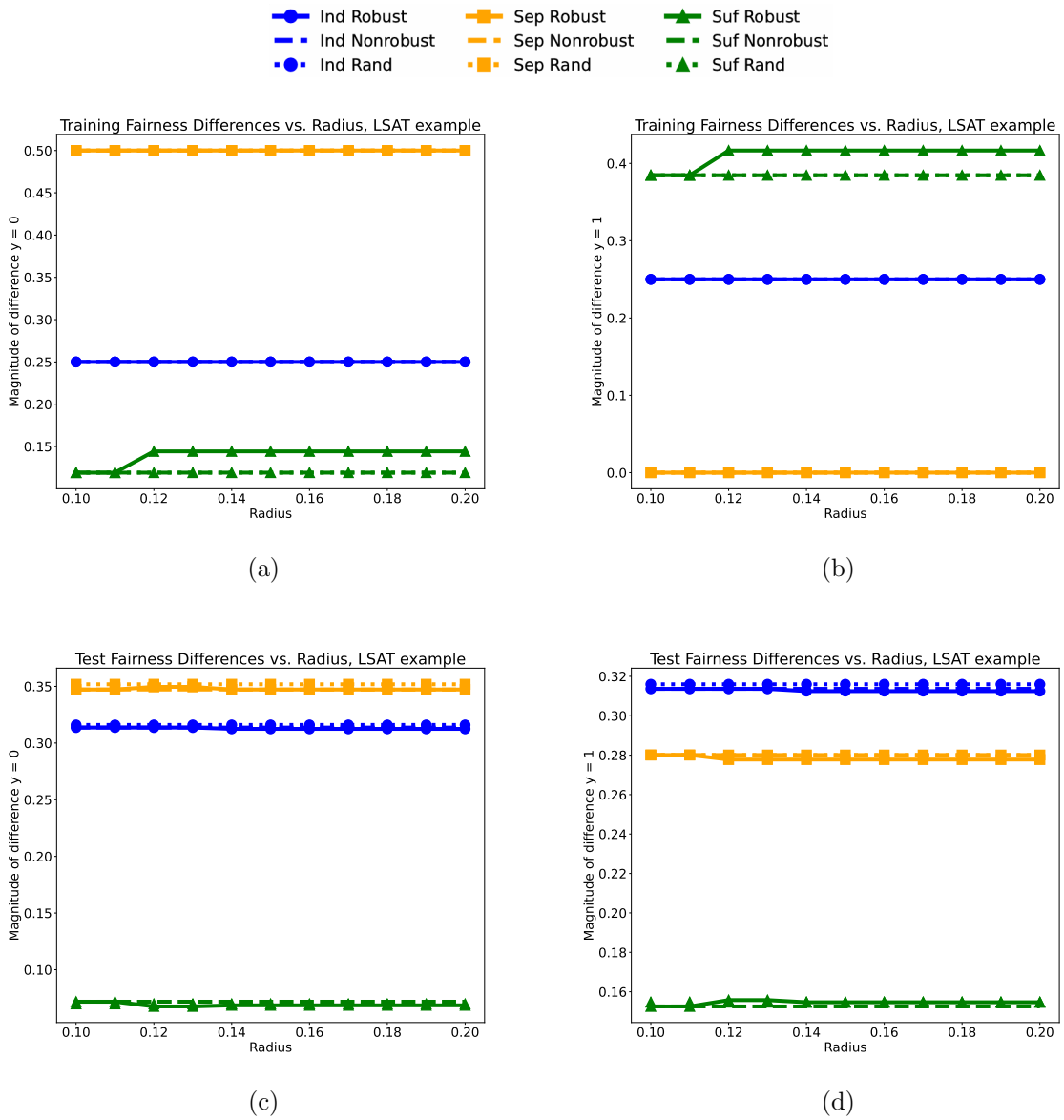
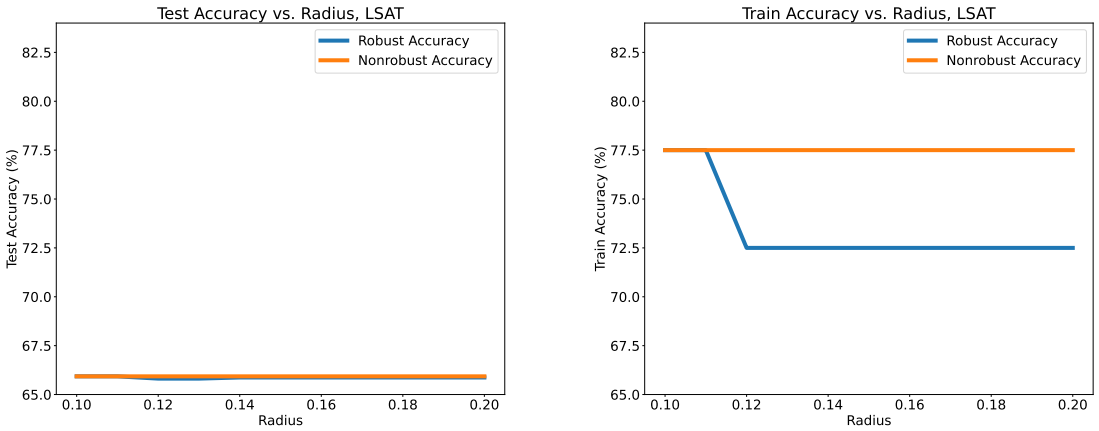


Figure 8: Fairness trends in the LSAT dataset for nonrobust and robust training with 11 different perturbation radii ranging from .1 to .2 with increments of .01. The experiment ran with 10 epochs and a learning rate of .01.

In the first row of Figure 8, we have two plots that show the fairness differences in the LSAT training data for y = 0 and y = 1. Similarly, in the second row, we have two plots that show the fairness differences in the LSAT testing data. As mentioned before, we desire to have the magnitude of the differences between our three metrics to be as small as possible.

Unlike the two previous examples, there is not a lot of fluctuation with the LSAT robust results. Surprisingly, robust optimization seems to not deviate from non-robust training at all. As we have seen, using a random perturbation radius yields fairness results that are very similar to the results of non-robust training. However, for this dataset, robust optimization performs very similarly to just picking a random perturbation radius. This is especially the case with the independence and separation metrics. For both the training and testing set, we can see that all of the blue lines and all of the yellow lines are on top of each other or very close together. The only two major deviations come from the sufficiency metric in Figure 8a and Figure 8b of the training dataset. (One explanation for this could be that the dataset becomes very small when we are defining the sensitive attribute to be white vs nonwhite.) For this specific example, we can see that robust optimization does not improve fairness. Looking at the accuracy trends, we can even argue that it performs worse since it loses accuracy when yielding the same fairness results.

4.4.2. LSAT Accuracy Trends.



4.5. MNIST. Annie: I'll look into this.

4.6. Efficiency Comparison. We also gathered time data to see if the trust region subproblem was performing faster than PGD on our datasets. Conveniently, in the running of our experiments, we were able to keep track of epoch time length internally. It was clear to us that epochs were a lot longer with robust training than with nonrobust training, which made it clear that solving the inner optimization problem took the bulk of the time in nonrobust training and that we could defensibly use epoch time as a relative gauge for how long it took to solve the inner optimization problem in a given epoch. Thus, on each of the three datasets above - the synthetic one, the LSAT one, and the Adult one - we computed, for each radius setting, the average epoch time length in running the trust region subproblem, PGD, and Random Perturbation. To compare the speed of the trust region subproblem and PGD, for each dataset, we examined the ratio of the average PGD epoch time to the average epoch time of the trust region subproblem method, looking at the minimum and maximum values of this ratio to get a range of how much faster the trust region subproblem was over PGD across all radii. The results are shown in the following tables.

Table 1: Average Epoch Times

	Radii	.10	.12	.14	.16	.18	.20
Unfair	PGD	2.680	3.597	4.228	4.486	5.061	5.080
	TRUST	1.945	1.875	1.806	1.891	1.742	1.752
	RAND	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

As we can see from Table 1, random perturbation is the fastest adversarial training method in all three examples. This is clearly due to the fact that it does not solve the optimization problem well; it only generates a random vector and rescales it, which is not very computationally expensive. It is also noteworthy that using the trust region subproblem method is clearly faster than using PGD in all three datasets. As can be seen in the table, for all of the six radii shown, training with the trust-region subproblem is on average between 1.377 (gray) and 2.904 (yellow) times faster than PGD; between 2.852 (gray) and 9.639 (yellow) times faster in LSAT; and between 8.497 (gray) and 31.407 (yellow) times faster than in Adult. All of the smallest factors of improvement (highlighted in gray) are greater than 1 in the table, suggesting that the trust region subproblem has a very clear advantage over PGD in computational efficiency. A few other observations are interesting to make from the ratio rows of the table. First, the factor of improvement that the trust region subproblem approach 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 there seems to be a pattern that the trust region subproblem has a greater edge over PGD the larger the dataset you are working with. The other pattern that seems to be noticeable is that the edge of the trust region subproblem approach over PGD appears, on the whole, to be greater with larger perturbation radii; it is telling that the minimum factor of improvement (gray) always occurs with the smallest radius, and the maximum factor of improvement (yellow) always occurs in the right half of the table with one of the three largest radii.

5. Conclusion. Due to the affine linear model of our setup, we were able to see improvement in fairness from using robust optimization. In the Unfair2D example, whenever there is an improvement, the gain is a significant reduction in differences in our unfairness metrics. 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 example).

Three trends remained the same across all three datasets: the accuracy of robust optimization decreases as the radius increases, the majority of the fairness metrics display a downward trend as the perturbation radius increases, and a precise solution to the optimization problems outperforms randomly selecting a solution that satisfies the constraint. Furthermore, we were able to quantify the fact that, with the help of HessQuik, using second-order information is much faster for solving optimization problems.

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 prevents any generalization to higher-dimension applications. All of our experiments were conducted using a linear loss function and a binary classifier. Additionally, our sensitive attribute was also defined to be binary. This motivates future exploration of extending our approach to multinomial classification and other fairness metrics. Finally, it may also be helpful to introduce a regularization term to our approach to penalize the terms that violate our fairness constraints.

Lastly, we want to talk about the limitations of our implementation. For PGD, we used an arbitrary step size instead of solving for what the step size should be. Additionally, we did not solve our optimization problems in parallel. One future approach would be to parallelize solving for the perturbation radius in our inner optimization problem so that we don't have to iterate through each data point.

Acknowledgments. This work was supported by the NS NSF award DMS-2051019 and was completed during the "Computational Mathematics for Data Science" REU/RET program in Summer 2023. We would like to thank Dr. Elizabeth Newman and the rest of the faculty members of the math department for their feedback and support.

Appendix A. An example appendix. Aenean tincidunt laoreet dui. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Integer ipsum lectus, fermentum ac, malesuada in, eleifend ut, lorem. Vivamus ipsum turpis, elementum vel, hendrerit ut, semper at, metus. Vivamus sapien tortor, eleifend id, dapibus in, egestas et, pede. Pellentesque faucibus. Praesent lorem neque, dignissim in, facilisis nec, hendrerit vel, odio. Nam at diam ac neque aliquet viverra. Morbi dapibus ligula sagittis magna. In lobortis. Donec aliquet ultricies libero. Nunc dictum vulputate purus. Morbi varius. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In tempor. Phasellus commodo porttitor magna. Curabitur vehicula odio vel dolor.

Lemma A.1. *Test Lemma.*

REFERENCES

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

[2] 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.

[3] M. Lohaus, M. Perrot, and U. V. Luxburg, *Too relaxed to be fair*, in Proceedings of the 37th International Conference on Machine Learning, H. D. III and A. Singh, eds., vol. 119 of Proceedings

of Machine Learning Research, PMLR, 13–18 Jul 2020, pp. 6360–6369, https://proceedings.mlr.press/v119/lohaus20a.html.

[4] 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.

[5] 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.

[6] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, NY, USA, 2e ed., 2006.

[7] 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.

[8] 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.