Post-Hoc Methods for Debiasing Neural Networks

Yash Savani
RealityEngines.AI
yash@realityengines.ai

Colin White
RealityEngines.AI
colin@realityengines.ai

Naveen Sundar Govindarajulu
RAIR Lab RPI
naveensundarg@gmail.com

Abstract

As deep learning models become tasked with more and more decisions that impact human lives, such as hiring, criminal recidivism, and loan repayment, bias is becoming a growing concern. This has led to dozens of definitions of fairness and numerous algorithmic techniques to improve the fairness of neural networks. Most debiasing algorithms require retraining a neural network from scratch, however, this is not feasible in many applications, especially when the model takes days to train or when the full training dataset is no longer available.

In this work, we present a study on post-hoc methods for debiasing neural networks. First we study the nature of the problem, showing that the difficulty of post-hoc debiasing is highly dependent on the initial conditions of the original model. Then we define three new fine-tuning techniques: random perturbation, layer-wise optimization, and adversarial fine-tuning. All three techniques work for any group fairness constraint. We give a comparison among three popular post-processing debiasing algorithms and our three proposed methods, across three datasets and three popular bias measures. Our algorithms outperform the existing post-processing techniques on average, and each of our algorithms perform best in certain settings. Our code is available at `https://github.com/realityengines/post_hoc_debiasing`.

1 Introduction

The last decade has seen a huge increase in applications of machine learning in a wide variety of domains such as credit scoring, fraud detection, hiring decisions, criminal recidivism, loan repayment, and so on [31, 6, 34, 2]. The outcome of these algorithms are impacting the lives of people more than ever. There are clear advantages in the automation of classification tasks, as machines can quickly process thousands of datapoints with many features. However, algorithms are susceptible to bias towards individuals or groups of people from a variety of sources [38, 35, 36]. For example, Correctional Offender Management Profiling for Alternative Sanctions (COMPAS) is a computer software which determines the risk of a defendant committing a future crime. United States judges consult the software to decide whether or not a defendant should be granted bail or pretrial release. It was found that this software is biased against African-Americans [15]. The need to address these issues is higher than ever [3, 37].

Motivated by these findings, the last few years has seen a huge growth in the area of fairness in machine learning. Dozens of formal definitions of fairness have been proposed [33], and many algorithmic techniques have been developed for debiasing according to these definitions [44]. While substantial progress has been made, the majority of techniques have been developed as pre-processing or in-processing methods. In other words, most techniques are developed to run before or during the training of the machine learning model, either as an add-on, or as a newly proposed algorithm.

Only a handful of debiasing methods run after the training has been completed, either as fine-tuning methods or post-processing methods [4]. However, as datasets become larger and training becomes more computationally intensive, especially in the case of neural networks, there is a growing need for debiasing algorithms which do not require retraining a model from scratch. Additionally, some applications may require debiasing an existing model without full access to the training dataset, due to regulatory requirements or privacy concerns. For example, this may be true any time the entities which build the model are different from the entities which deploy the model. Furthermore, most previously proposed post-processing methods have been designed for just one fairness measure. Due to the diversity in fairness definitions, and since reduction of more than one fairness measure may be difficult [10], there is a growing need for algorithms which can handle any group fairness constraint.

In this work, we present a formal study of post-hoc methods for debiasing neural networks. A post-hoc method is defined as an algorithm which has access to a trained model and a validation dataset, and either fine-tunes the model or performs post-processing on the model predictions. We start by showing that the difficulty of post-hoc debiasing is highly dependent on the initial conditions of the original model. In particular, given a neural network trained to optimize accuracy, the variance in the amount of bias of the trained model is much higher than the variance in the accuracy, with respect to the random seed used for initializing the weights of the original model. Therefore, even the initial random seed can substantially change the amount of bias present in a neural network.

Next, we present three new optimization-based techniques for post-hoc debiasing of neural networks, each of which work for any group fairness measure. Each technique takes as input an objective function, which can be chosen to trade off a fairness measure with model accuracy. We define a simple algorithm, random perturbation, which iteratively adds multiplicative noise to the weights of the neural network and then thresholds the output probabilities to minimize the objective function. Our second technique is a layer-wise optimization algorithm. In this approach, we iteratively choose a layer of the neural network and use gradient-boosted regression trees to optimize the weights of the chosen layer with respect to the objective function. Our last technique is an adversarial fine-tuning algorithm. Adversarial training is a powerful debiasing method because training a critic (discriminator) to predict bias effectively makes the objective function differentiable, enabling the use of first-order optimization techniques such as gradient descent. This has recently been proposed as an in-processing method for debiasing [45]. We show that using an adversarial model to fine-tune the trained neural network is a viable post-hoc technique.

We compare the three above techniques with three post-processing algorithms from prior work: reject option classification [23], equalized odds post-processing [21], and calibrated equalized odds post-processing [39]. We run experiments with three popular fairness datasets and three popular fairness definitions. We show that certain algorithms are useful in certain scenarios. For example, the random perturbation algorithm is a strong post-hoc debiasing baseline. The adversarial fine-tuning method is more powerful for debiasing larger models, but it is more computationally intensive and may require hyperparameter tuning. The layer-wise fine-tuning algorithm may work well on models in which the bias is concentrated in one layer.

Fairness research (and machine learning research as a whole) has seen a huge increase in popularity, and recent papers have highlighted the need for fair and reproducible results [42, 4]. To facilitate best practices, we run our experiments on the AIF360 toolkit [4] and open source all of our code.

Our contributions. We summarize our main contributions below.

- We study the nature of post-hoc techniques for debiasing neural networks, showing that the

problem is sensitive to the initial conditions of the original model.

- We present three measure agnostic, fine-tuning algorithms for post-hoc debiasing: random perturbation, layer-wise optimization, and adversarial fine-tuning. Our algorithms outperform all existing post-processing techniques on average.
- We conduct a study of post-hoc techniques for debiasing neural networks, testing six different algorithms across three datasets and with three different fairness measures.

2 Related Work

Debiasing overview. There is a surging body of research on bias and fairness in machine learning. There are dozens of types of bias that can arise [29], and dozens of formal definitions of fairness have been proposed [33]. Popular definitions include statistical parity/demographic parity [14, 25], equal opportunity (a subset of equalized odds) [20], and average absolute odds [4]. Many bias mitigation techniques have been proposed, which generally fall into three categories: pre-processing, in-processing, and post-processsing. Post-processing debiasing techniques are performed on a pretrained model and do not require access to the full training set. Therefore, these techniques are useful in a variety of settings in which retraining is costly or impossible due to computational costs or data limitations.

Post-processing methods Most prior work on post-processing techniques use label-flipping methods such as randomly flipping labels until the true/false negative rates are equal, or flipping labels in a critical region of predicted probabilities near 0.5 [20, 39, 23]. Currently, most of these techniques have only been established for specific fairness measures. For a full overview, see [4, 44]. See Section 6 for brief descriptions of three post-processing debiasing techniques

Hyperparameter optimization for fairness There is a variety of work on in-processing debiasing algorithms which are similar in spirit to our optimization methods. We mention a few of them here. However, none of these explicitly present a post-hoc debiasing algorithm. Recently, a meta-algorithm was developed for in-processing debiasing by reducing many fairness measures to convex problems [8]. Another work treats debiasing as an empirical risk minimization problem [12]. Yet another work adds the fairness constraints as regularizers in the machine learning models [5]. Other prior work has used hyperparameter optimization to select hyperparameters for training models to exhibit less bias [9], but this approach repeatedly retrains the full model with different hyperparameters. Bias reduction has also been framed as a pre-processing convex optimization problem [7].

There is also prior work using adversarial learning to debias algorithms [45]. To the best of our knowledge, no prior work has designed a post-hoc algorithm using adversarial learning.

3 Broader Impact

Deep learning algorithms are more prevalent than ever before. The technology is becoming more and more integrated into society, and is used in high-stakes applications such as criminal recidivism, loan repayment, and hiring decisions [31, 6, 34, 2]. It is also becoming increasingly more evident

that many of these algorithms are biased from various sources [38, 35, 36]. Using technology for life-changing events which make prejudiced decisions will only deepen the divides that exist in society, and the need to address these issues is higher than ever [3, 37].

Our work seeks to decrease the negative effects that biased deep learning algorithms have on society. Post-hoc methods, which work for any group fairness measure, will be applicable to large existing deep learning models, since the networks need not be retrained from scratch. Furthermore, we present simple techniques (random perturbation) as well as more complex and strong techniques (adversarial fine-tuning). Since we study the nature of post-hoc debiasing and present a study comparing prior work to our algorithms, our work may facilitate future work in post-hoc debiasing techniques.

Impact on bias in judicial applications We briefly discuss how post-hoc methods for debiasing could help in judicial settings. Some machine learning algorithms which are prejudiced have been used in judicial applications in the past [41, 19]. Studies and investigations have found that many of the algorithms have some form of bias [26]. Moreover, different entities using the same model might prefer to use different fairness measures and some of these measures might be incompatible [11]. Generally, the entities that build and use the applications are not the same. Therefore, due to legal and licensing issues, the entity using the application may not have access to the training dataset. This precludes the use of pre-processing and in-processing methods for debiasing. The entity using the model usually has its own dataset available (e.g. a local court tracking their recidivism rates). This makes post-hoc processing the only viable method for debiasing.

4 Preliminaries

In this section, we give notation and definitions used throughout the paper. Given a dataset split into three parts, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{valid}}$, $\mathcal{D}_{\text{test}}$, let $(\boldsymbol{x}_i, Y_i)$ denote one datapoint, where $\boldsymbol{x}_i \in \mathbb{R}^d$ contains d features including one binary protected attribute A (e.g., identifying as female or not identifying as female), and $Y_i \in \{0, 1\}$ is the label. Denote the value of the protected feature for $\boldsymbol{x}_i$ as a_i. We denote a trained neural network by a function $f_\theta : \mathbb{R}^d \to [0, 1]$, where θ denotes the trained weights. We often denote $f_\theta(\boldsymbol{x}_i) = \hat{Y}_i$, the output predicted probability for datapoint $\boldsymbol{x}_i$. Finally, we refer to a set of labels in a dataset $\mathcal{D}$ as $\mathcal{Y}$.

Fairness measures. We now give an overview of group fairness measures used in this work. Given a dataset $\mathcal{D}$ with labels $\mathcal{Y}$, protected attribute A, and a set of predictions $\hat{\mathcal{Y}} = \{f_\theta(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{D}\}$ from some neural network f_θ, we define the true positive and false positive rates as

$$TPR_{A=a}(\mathcal{D}, \hat{\mathcal{Y}}) = \frac{\left|\{i \mid \hat{Y}_i = Y_i = 1, a_i = a\}\right|}{\left|\{i \mid \hat{Y}_i = Y_i = 1\}\right|} = P_{(\boldsymbol{x}_i, Y_i) \in \mathcal{D}}(\hat{Y}_i = 1 \mid a_i = a, Y_i = 1),$$

$$FPR_{A=a}(\mathcal{D}, \hat{\mathcal{Y}}) = \frac{\left|\{i \mid \hat{Y}_i = 1, Y_i = 0, a_i = a\}\right|}{\left|\{i \mid \hat{Y}_i = 1, Y_i = 0\}\right|} = P_{(\boldsymbol{x}_i, Y_i) \in \mathcal{D}}(\hat{Y}_i = 1 \mid a_i = a, Y_i = 0),$$

Statistical Parity Difference (SPD), or demographic parity difference [14, 25], measures the difference in the probability of a positive outcome between the protected and unprotected groups. Formally,

$$SPD(\mathcal{D}, \hat{\mathcal{Y}}, A) = P_{(\boldsymbol{x}_i, Y_i) \in \mathcal{D}}(\hat{Y}_i = 1 \mid a_i = 0) - P_{(\boldsymbol{x}_i, Y_i) \in \mathcal{D}}(\hat{Y}_i = 1 \mid a_i = 1).$$

Equal opportunity difference (EOD) [20] measures the difference in TPR for the protected and unprotected groups. Equal opportunity is identical to *equalized odds* in the case where the protected feature and labels are binary. Formally, we have

$$EOD(\mathcal{D}, \hat{\mathcal{Y}}, A) = TPR_{A=0}(\mathcal{D}, \hat{\mathcal{Y}}) - TPR_{A=1}(\mathcal{D}, \hat{\mathcal{Y}}).$$

Average Odds Difference (AOD) [4] is defined as the average of the difference in the false positive rates and true positive rates for unprivileged and privileged groups. Formally,

$$AOD(\mathcal{D}, \hat{\mathcal{Y}}, A) = \frac{(FPR_{A=0}(\mathcal{D}, \hat{\mathcal{Y}}) - FPR_{A=1}(\mathcal{D}, \hat{\mathcal{Y}})) + (TPR_{A=0}(\mathcal{D}, \hat{\mathcal{Y}}) - TPR_{A=1}(\mathcal{D}, \hat{\mathcal{Y}}))}{2}.$$

Optimization techniques. Zeroth order (non-differentiable) optimization is used when the objective function is not differentiable (as is the case for most definitions of group fairness). This is also called black-box optimization. Given an input space W and an objective function μ, zeroth order optimization seeks to compute $w^* = \arg\min_{w \in W} \mu(w)$. Leading methods for zeroth order optimization when function queries are expensive (such as optimizing a deep network) include gradient-boosted regression trees (GBRT) [17, 28] and Bayesian optimization (BO) [40, 16, 43], however BO struggles with high-dimensional data.

In contrast, first-order optimization is used when it is possible to take the derivative of the objective function. For example, gradient descent is a first-order optimization technique.

5 Methodology

In this section, we describe three new fine-tuning techniques for debiasing neural networks. First we give more notation and formally define the different types of debiasing algorithms.

Given a neural network f_θ, we sometimes drop the subscript θ when it is clear from context. We denote the last layer of f by $f^{(\ell)}$, and we assume that $f = f^{(\ell)} \circ f'$, where f' is all but the last layer of the neural network. Our layer-wise optimization algorithm assumes that f is feed-forward, that is, $f = f^{(\ell)} \circ \cdots \circ f^{(1)}$ for functions $f^{(1)}$, $f^{(2)}, \ldots, f^{(\ell)}$. The performance of the model is given by a performance measure ρ. For a set of points $\mathcal{D}$, given the set of true labels $\mathcal{Y}$ and the set of predicted labels $\hat{\mathcal{Y}} = \{f(\boldsymbol{x}_i) \mid (\boldsymbol{x}_i, Y_i) \in \mathcal{D}'\}$, the performance is $\rho(\mathcal{Y}, \hat{\mathcal{Y}}) \in [0, 1]$. Common performance measures include accuracy, precision, recall, or AUC ROC (area under the ROC curve). We also define a bias measure μ, given as $\mu(\mathcal{D}, \hat{\mathcal{Y}}, A) \in [0, 1]$, such as one defined in Section 4.

The goal of any debiasing algorithm is to minimize the bias μ, without sacrificing performance ρ too much. Many prior works have observed that fairness comes at the price of accuracy for many datasets, even when using large models such as deep networks [4, 44, 10], which means it is often not possible to achieve zero bias without significantly lowering accuracy. Therefore, a common technique is to minimize an objective function such as the following.

$$\phi_{\mu,\rho}(\mathcal{D}, \hat{\mathcal{Y}}, A) = \lambda \cdot |\mu(\mathcal{D}, \hat{\mathcal{Y}}, A)| + (1 - \lambda)(1 - \rho(\mathcal{Y}, \hat{\mathcal{Y}})). \tag{1}$$

In the expression, λ is a parameter in $[0, 1]$ which can be tuned based on the desired bias or based on the level of bias in the original model.

An in-processing debiasing algorithm takes as input the training and validation datasets and outputs a model f which seeks to minimize $\phi_{\mu,\rho}$. A fine-tuning algorithm takes in the validation dataset and a trained model f with weights θ (typically f was trained to optimize the performance ρ), and outputs fine-tuned weights θ' such that $f_{\theta'}$ minimizes the objective $\phi_{\mu,\rho}$. A post-processing debiasing algorithm takes as input the validation dataset as well as a set of predictions $\hat{\mathcal{Y}}$ on the validation dataset (typically coming from a model f which was optimized for ρ), and outputs a post-processing function $h : \{0, 1\} \to \{0, 1\}$ which performs post-processing on predictions so that the final predictions optimize $\phi_{\mu,\rho}$. Note that fine-tuning and post-processing debiasing algorithms are useful in different settings. Post-processing algorithms are useful when there is no access to the original model. Fine-tuning algorithms are useful when there is access to the original model, or when the prediction is over a continuous feature. Now we present three fine-tuning techniques.

Random perturbation. Our first algorithm is a simple iterative random procedure, *random perturbation*. In every iteration, each weight in the neural network is multiplied by a Gaussian random variable with mean 1 and standard deviation 0.1. In case the model f outputs probabilities, we find the threshold τ such that $\hat{\mathcal{Y}}_\tau = \{\mathbb{I}\{\hat{Y} > \tau\}\}_{\hat{Y} \in \hat{\mathcal{Y}}}$ minimizes $\phi_{\mu,\rho}(\mathcal{Y}, \hat{\mathcal{Y}}_\tau, A)$. We run T iterations and output the perturbed weights which minimize $\phi_{\mu,\rho}$ on the validation set. See Algorithm 1. We show in the next section that despite its simplicity, this model performs well on many datasets and fairness measures, and therefore we recommend this algorithm as a baseline in future post-hoc debiasing applications. A natural follow-up question is whether we can do even better by using an optimization algorithm instead of random search. This is the motivation for our next approach.

Algorithm 1 Random Perturbation

Input: Trained model f with weights θ, validation dataset $\mathcal{D}_{\text{valid}}$, objective $\phi_{\mu,\rho}$, parameter T
Set $\theta^* = \emptyset$, $\text{val}^* = \infty$, and $\tau^* = 0$
for $i = 1$ to T **do**
  Sample $q_j \sim \mathcal{N}(1, 0.1)$ for all $j \in \{1, 2, ..., |\theta|\}$
  $\theta'_j = \theta_j \cdot q_j$
  Select threshold $\tau \in [0, 1]$ which minimizes the objective $\phi_{\mu,\rho}$ on the validation set
  Set $\text{val} = \phi_{\mu,\rho}(\mathcal{D}_{\text{valid}}, \{\mathbb{I}\{f_{\theta'}(\boldsymbol{x}) > \tau\} \mid (\boldsymbol{x}, Y) \in \mathcal{D}_{\text{valid}}\}, A)$
  If $\text{val} < \text{val}^*$, set $\text{val}^* = \text{val}$, $\theta^* = \theta'$ and $\tau^* = \tau$
end for
Output: θ^*, τ^*

Layer-wise optimization. Our next method fine-tunes the model by debiasing individual layers using zeroth order optimization. Intuitively, an optimization procedure will be much more effective than random perturbations, but it is computationally expensive and does not scale as well, so we can only run optimization on individual layers. Given a model, assume the model can be decomposed into several functions $f = f^{(\ell)} \circ \cdots \circ f^{(1)}$ For example, a feed-forward neural network with ℓ layers can be decomposed in this way. We denote the trained weights of each component by $\theta_1, \ldots, \theta_\ell$, respectively. Now assume that we have access to a zeroth order optimizer $\mathcal{A}$, which takes as input a model $f = f^{(\ell)} \circ \cdots \circ f^{(1)}$, weights $\theta = \theta_1, \ldots, \theta_\ell$, dataset $\mathcal{D}_{\text{valid}}$, and an index i. The optimizer

returns weights θ'_i, optimized with respect to to $\phi_{\mu,\rho}$. In Algorithm 2, we set the optimizer to be gradient-boosted regression trees (GBRT) [17, 28], a leading technique for black box optimization which converts shallow regression trees into strong learners. GBRT iteratively constructs a posterior predictive model using the weights to make prediction and uncertainty estimates for each potential set of weights θ'. To trade off exploration and exploitation, the next set of weights to try is chosen using lower confidence bounds (LCB), a popular acquisition function (e.g., [22]). Formally, $\phi_{\text{LCB}}(\theta') = \hat{\theta}' - \beta\hat{\sigma}$, in which we assume our model's posterior predictive density follows a normal distribution with mean $\hat{\theta}'$ and standard deviation $\hat{\sigma}$. β is a tradeoff parameter that can be tuned. See Algorithm 2. Note that this algorithm can be easily generalized to optimize multiple layers at once, but this comes at the price of runtime. For example, running GBRT on the entire neural network would be strictly more powerful than the random permutation algorithm but is prohibitively expensive.

Algorithm 2 Layer-wise optimization

Input: Trained model $f = f^{(\ell)} \circ \ldots \circ f^{(1)}$ with weights $\theta_1, \ldots, \theta_\ell$, objective $\phi_{\mu,\rho}$, optimizer $\mathcal{A}$
Set $\theta^* = \emptyset$, $\text{val}^* = \infty$, and $\tau^* = 0$
for $i = 1$ to ℓ **do**
    Run optimizer $\mathcal{A}$ to optimize weights θ_i to θ'_i with respect to $\phi_{\mu,\rho}$.
    Select threshold $\tau \in [0, 1]$ which minimizes objective $\phi_{\mu,\rho}$
    Set $\text{val} = \phi_{\mu,\rho}(\mathcal{D}_{\text{valid}}, \{\mathbb{I}\{f_{\theta'}(\boldsymbol{x}) > \tau\} \mid (\boldsymbol{x}, Y) \in \mathcal{D}_{\text{valid}}\}, A)$, where $\theta' = \{\theta_1, ..., \theta'_i, ..., \theta_\ell\}$
    If $\text{val} < \text{val}^*$ set $\text{val}^* = \text{val}$, and $\theta^* = \theta'$.
end for
Output: θ^*, τ^*

Adversarial fine-tuning. The previous two methods rely on zeroth order optimization techniques because most group fairness measures such as statistical parity difference and equalized odds are non-differentiable. Our last technique casts the problem of debiasing as first-order optimization by using adversarial learning. The idea behind the adversarial method is that we train a critic model to predict the amount of bias in a minibatch. We sample the datapoints in a minibatch randomly and with replacement. This statistical bootstrapping approach to creating a minibatch means that if the critic can predict the bias in a minibatch accurately, then it can predict the bias in the model with respect to the validation set reasonably well. Therefore, the critic effectively acts as a differentiable proxy for bias, which makes it possible to debias the original model using gradient descent.

The adversarial algorithm works by alternately iterating between training the critic model g using the predictions from f, and fine-tuning the predictive model f with respect to $\phi_{\mu,\rho}$ using the bias proxy $\hat{\mu}$ from g. Note that the first layer in g concatenates the minibatch and returns a single number that estimates the bias of the minibatch as the final output. See Algorithm 3. Note that BCELoss denotes the standard binary cross-entropy loss.

6 Experiments

In this section, we experimentally evaluate the techniques laid out in Section 5 compared to baselines, on three datasets and with multiple fairness measures. To promote reproducibility, we release our code at `https://github.com/realityengines/post_hoc_debiasing` and we use popular datasets

Algorithm 3 Adversarial Fine-Tuning

Input: Trained model $f = f_\ell \circ f'$ with weights θ, validation dataset $\mathcal{D}_{\text{valid}}$, objective $\phi_{\mu,\rho}$ parameters $\lambda,\ m,\ m',\ T$
Set g as the critic model with weights θ'.
for $i = 0$ to n **do**
  for $j = 0$ to m **do**
    Sample a minibatch $(\boldsymbol{X}_k, \boldsymbol{\mathcal{Y}}_k)$ with replacement from $\mathcal{D}_{\text{valid}}$
    Evaluate the bias in the minibatch, $\hat{\mu} \leftarrow \mu((\boldsymbol{X}_k, \boldsymbol{\mathcal{Y}}_k), f(\boldsymbol{X}_k))$.
    Update the critic model g by updating its stochastic gradient

$$\nabla_{\theta'}(\hat{\mu} - (g \circ f')(\boldsymbol{X}_k))^2$$

  end for
  for $j = 0$ to m' **do**
    Sample a minibatch $(\boldsymbol{X}_k, \boldsymbol{\mathcal{Y}}_k)$ with replacement from $\mathcal{D}_{\text{valid}}$
    Update the original model by updating its stochastic gradient

$$\nabla_\theta \left[(1-\lambda) \cdot (g \circ f')(\boldsymbol{X}_k) + \lambda \cdot \text{BCELoss}(\boldsymbol{\mathcal{Y}}_k, f(\boldsymbol{X}_k))\right]$$

  end for
  Select threshold $\tau \in [0, 1]$ that minimizes the objective $\phi_{\mu,\rho}$
end for
Output: Debiased model f, threshold τ

from the AIF360 toolkit [4]. Each dataset contains one or more binary protected features(s) and a binary label. We briefly describe them below.

The COMPAS dataset is a commonly used dataset in fairness research, consisting of over 10,000 defendants with 402 features [15]. The goal is to predict the recidivism likelihood for an individual [1]. We run separate experiments using *race* and also *sex* as protected attributes. The Adult Census Income (ACI) dataset is a binary classification dataset from the 1994 USA Census bureau database in which the goal is to predict whether a person earns above \$50,000 [13]. There are over 40,000 data points with 15 features. We use *sex* as the protected attribute. The Bank Marketing (BM) dataset is from the phone marketing campaign of a Portuguese bank. There are over 48,000 datapoints consisting of 17 categorical and quantitative features. The goal is to predict whether a customer will subscribe to a product [30]. The protected feature is whether or not the customer is older than 25.

The need for neural networks. First, we run a quick experiment to demonstrate the need for neural networks on the above datasets. Deep learning has become a very popular approach in the field of machine learning [27], however, for tabular datasets with fewer than 20 features, it is worth checking whether logistic regression or random forest techniques perform as well as neural networks [32]. We construct a neural network with 10 fully-connected layers, BatchNorm for regularization, and a dropout rate of 0.2, and we compare this to logistic regression and a random forest model on the ACI dataset. We see that a neural network achieves accuracy and area under the receiver operating characteristic curve (AUC ROC) scores which are 2% higher than the other models. See Appendix A for the full results. Therefore, for the rest of this section, we focus on using

Table 1: Bias and accuracy of a neural network.

	AOD	EOD	SPD	accuracy
ACI (sex)	-0.084 ± 0.012	-0.082 ± 0.017	-0.198 ± 0.011	0.855±0.002
BM (age)	0.011 ± 0.027	-0.009 ± 0.051	0.047 ± 0.015	0.901±0.002
COMPAS (race)	0.138 ± 0.017	0.194 ± 0.027	0.168 ± 0.016	0.669±0.006

neural networks.

Bias sensitivity to initial model conditions. Next, we run experiments to compute the amount of variance in the bias scores of the initial models. Neural networks have a huge number of local minima. Hyperparameters such as the optimizer and learning rate, and even the initial random seed, cause the model to converge to different local minima [27]. Techniques such as the Adam optimizer and early stopping with patience have been designed to allow neural networks to consistently reach local minima with high accuracies [24, 18]. However, there is no guarantee on the amount of bias. In particular, the local minima found by neural networks may have large differences in the amount of bias, and therefore, there may be very high variance on the amount of bias exhibited by neural networks just because of the random seed. Every local optima has a different set of weights. If the weights of the model at a specific local optimum rely heavily on the protected feature, removing the bias from such a model by updating the weights is harder than removing the bias from a model whose weights do not rely on the protected feature as heavily. Table 1 shows the mean and the standard deviation of three fairness measures, as well as accuracy, for training a neural network with 10 different initial random seeds, across three datasets. We see that the standard deviation of the bias score is an order of magnitude higher than the standard deviation of the accuracy. In Appendix A, we plot the contribution of each individual weight to the bias score, for a neural network. We show that the contribution of the weights to the bias score are sensitive to the initial random seed.

6.1 Post-hoc debiasing experiments

Now we present our main experimental study by comparing our three post-hoc debiasing methods to three baseline methods on three datasets and with three fairness measures. Note that we do not compare to any in-processing debiasing algorithms, because these algorithms require the entire training set, yet all post-hoc methods only use the validation set. We briefly describe the baseline post-processing algorithms that we tested.

The *reject option classification* post-processing algorithm [23] defines a critical region of points in the protected group whose predicted probability is near 0.5, and flips these labels. This algorithm is designed to minimize statistical parity difference. The *equalized odds* post-processing algorithm [20] defines a convex hull based on the bias rates of different groups, and then flips the label of data points that fall inside the convex hull. This algorithm is designed to minimize equal opportunity difference. The *Calibrated equalized odds* post-processing algorithm [39] defines a base rate of bias for each group, and then adds randomness based on the group into the classifier until the bias rates converge. This algorithm is designed to minimize equal opportunity difference. For all algorithms, we use the implementations in the AIF360 repository [4].

Our initial model consists of a feed-forward neural network with 10 fully-connected layers of size 32, with a BatchNorm layer between each fully-conntected layer, and a dropout fraction of 0.2. The model is trained with the Adam optimizer and an early-stopping patience of 100 epochs. The loss function is the binary cross-entropy loss. We use the validation data as the input for the post-hoc debiasing methods. The three post-hoc methods are set to optimize Equation 1 with $\lambda = 0.75$. We run each post-hoc method on 10 neural networks initialized with different random seeds. In Figure 1 we plot the objective function (plots 1-3) and accuracy + bias (plots 4-6) for all post-hoc debiasing algorithms, on all datasets and all fairness measures. Note that we ran separate experiments on COMPAS with *race* as the protected feature, and with *gender* as the protected feature. Note that since the three post-processing baselines are only set up to minimize a specific fairness measure, there is only a fair comparison on their respective measures.

Next, we vary the hyperparameters of the initial neural network. We run experiments on three additional neural networks: *(1)* dropout probability at 0.5 instead of 0.2, *(2)* width of each layer set to 64 instead of 32, *(3)* number of layers set to 20 instead of 10. We run these experiments on the ACI and BM datsets with SPD, for all post-hoc algorithms except for layerwise-optimization. We run 10 trials of each post-hoc algorithm on each neural network. See Figure 1 (plots 7-8).

Discussion. We see that the three fine-tuning methods significantly outperform the baseline methods, sometimes even on the fairness metric for which the baseline was designed. We note that there are two caveats. First, the three fine-tuning methods had access to the objective function in Equation 1, while the post-processing methods are only designed to minimize their respective fairness measures. However, as seen in Figure 1, sometimes the fine-tuning methods simultaneously achieve higher accuracy and lower bias compared to the post-processing methods, making the fine-tuning methods Pareto-optimal. Second, fine-tuning methods are more powerful than post-processing methods, since post-processing methods do not modify the weights of the original model, although it comes at the price of computation time (See Table 2). Post-processing methods are more appropriate when the model weights are unavailable or when computation time is constrained, and fine-tuning methods are more appropriate when higher performance is desired. We see that random perturbation is a strong fine-tuning technique, performing the best in many settings. Layer-wise optimization performs well in some settings, but is sometimes susceptible to the initial conditions of the original model which makes intuitive sense given the discussion earlier in this section on bias sensitivity to initial model conditions. The adversarial fine-tuning algorithm performs especially well when the dropout probability is higher and when the initial neural network is larger. This is likely due to the fact that adversarial fine-tuning is the most powerful technique (training a neural network as a subroutine).

7 Conclusion

In this work, we present a study on post-hoc methods for debiasing neural networks. We define three new measure-agnostic fine-tuning algorithms for debiasing neural networks: random perturbation, adversarial fine-tuning, and layer-wise optimization. First we show that the amount of bias is sensitive to the initial conditions of the original neural network. Then we give an extensive study of post-hoc debiasing by comparing our three new algorithms with three baseline post-processing algorithms on three popular fairness datasets and with three popular fairness measures. We show that each fine-tuning algorithm performs well for different datasets and different fairness metrics.

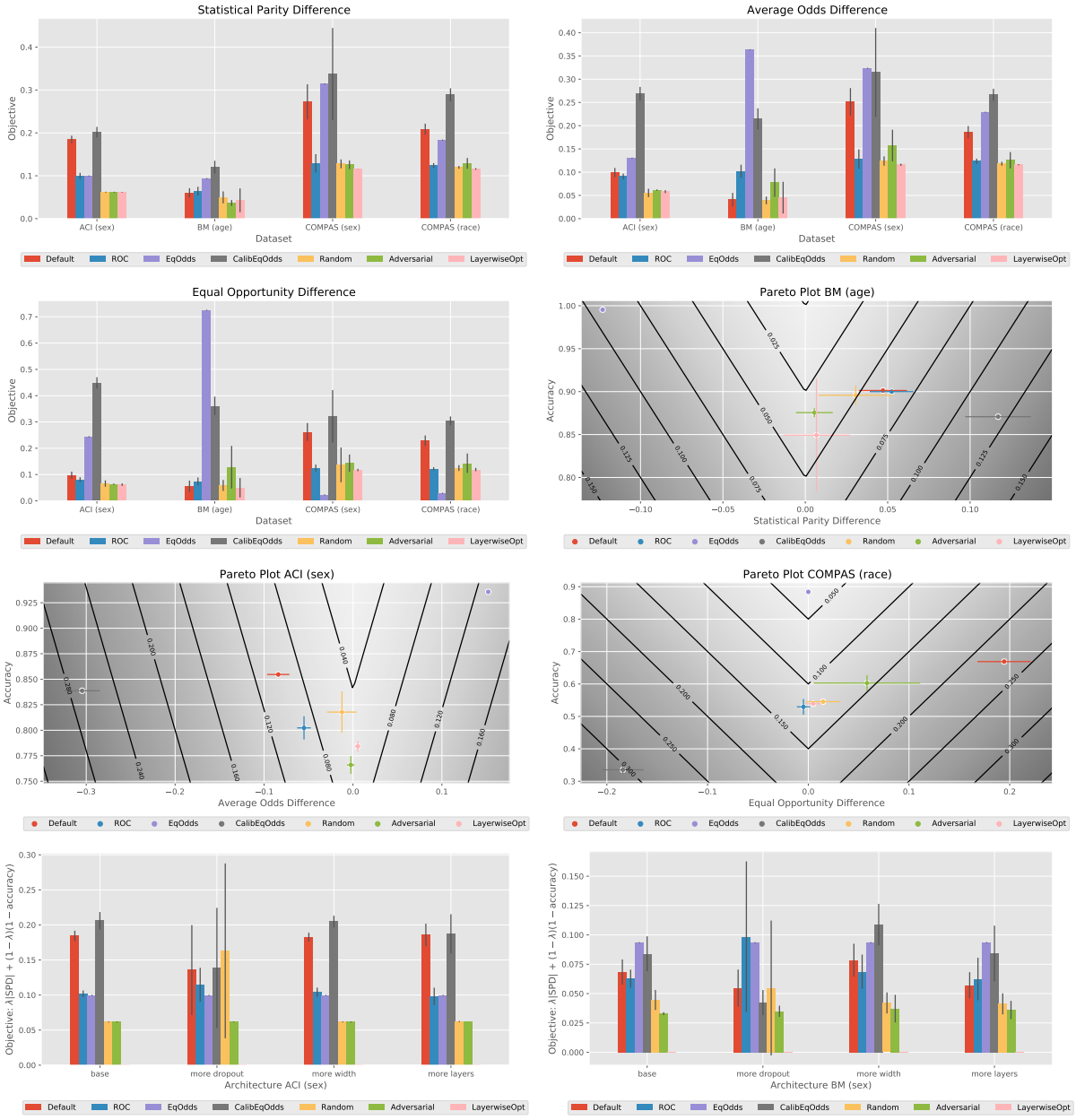

Figure 1: Results for post-hoc experiments. Plots 1-3: performance of six post-hoc debiasing algorithms across three datasets and three bias measures, plotted with respect to the objective function in Equation 1 which trades off bias with accuracy (a lower score is better). Plots 4-6: plots of the bias and accuracy of the above experiments. The objective function is shown as black contour lines. Plots 7-8: additional experiments which change the hyperparameters of the original neural network.

Table 2: Runtime for every post-hoc algorithm for every dataset in seconds

	ACI (sex)	BM (age)	COMPAS (sex)	COMPAS (race)
ROC	29.836	20.637	9.979	10.532
EqOdds	0.015	0.012	0.011	0.011
CalibEqOdds	0.144	0.064	0.049	0.054
Random	156.848	113.529	61.937	63.540
Adversarial	32.889	36.128	36.156	34.432
LayerwiseOpt	186.480	146.760	79.800	79.800

Acknowledgements

We thank Murali Narayanaswamy and anonymous reviewers for their help with this project.

References

[1] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. *ProPublica, May*, 23:2016, 2016.

[2] Solon Barocas, Moritz Hardt, and Arvind Narayanan. Fairness in machine learning. *NIPS Tutorial*, 2017.

[3] Jason Bellamy. Message from president dunn on racism and systemic inequality in america. 2020.

[4] Rachel KE Bellamy, Kuntal Dey, Michael Hind, Samuel C Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, et al. Ai fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. *arXiv preprint arXiv:1810.01943*, 2018.

[5] Richard Berk, Hoda Heidari, Shahin Jabbari, Matthew Joseph, Michael Kearns, Jamie Morgenstern, Seth Neel, and Aaron Roth. A convex framework for fair regression. *arXiv preprint arXiv:1706.02409*, 2017.

[6] Miranda Bogen and Aaron Rieke. Help wanted: An examination of hiring algorithms, equity, and bias, 2018.

[7] Flavio Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. Optimized pre-processing for discrimination prevention. In *Advances in Neural Information Processing Systems*, pages 3992–4001, 2017.

[8] L Elisa Celis, Lingxiao Huang, Vijay Keswani, and Nisheeth K Vishnoi. Classification with fairness constraints: A meta-algorithm with provable guarantees. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 319–328, 2019.

[9] Joymallya Chakraborty, Tianpei Xia, Fahmid M. Fahid, and Tim Menzies. Software engineering for fairness: A case study with hyperparameter optimization. *CoRR*, abs/1905.05786, 2019.

[10] Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big data*, 5(2):153–163, 2017.

[11] Alexandra Chouldechova and Aaron Roth. A snapshot of the frontiers of fairness in machine learning. *Communications of the ACM*, 63(5):82–89, 2020.

[12] Michele Donini, Luca Oneto, Shai Ben-David, John S Shawe-Taylor, and Massimiliano Pontil. Empirical risk minimization under fairness constraints. In *Advances in Neural Information Processing Systems*, pages 2791–2801, 2018.

[13] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[14] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, 2012.

[15] Anthony W Flores, Kristin Bechtel, and Christopher T Lowenkamp. False positives, false negatives, and false analyses: A rejoinder to machine bias: There's software used across the country to predict future criminals. and it's biased against blacks. *Fed. Probation*, 2016.

[16] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[17] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[19] Jamie Grace. Machine learning technologies and their inherent human rights issues in criminal justice contexts. *Available at SSRN 3487454*, 2019.

[20] Moritz Hardt, Eric Price, ecprice, and Nati Srebro. Equality of opportunity in supervised learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3315–3323. Curran Associates, Inc., 2016.

[21] Moritz Hardt, Eric Price, Nati Srebro, et al. Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, pages 3315–3323, 2016.

[22] Matthew Joseph, Michael Kearns, Jamie H Morgenstern, and Aaron Roth. Fairness in learning: Classic and contextual bandits. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, pages 325–333, 2016.

[23] Faisal Kamiran, Asim Karim, and Xiangliang Zhang. Decision theory for discrimination-aware classification. In *2012 IEEE 12th International Conference on Data Mining*, pages 924–929. IEEE, 2012.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. In *Advances in Neural Information Processing Systems*, pages 4066–4076, 2017.

[26] Jeff Larson, Surya Mattu, Lauren Kirchner, and Julia Angwin. How we analyzed the compas recidivism algorithm. *ProPublica (5 2016)*, 9, 2016.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[28] Llew Mason, Jonathan Baxter, Peter L Bartlett, and Marcus R Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

[29] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *arXiv preprint arXiv:1908.09635*, 2019.

[30] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[31] Amitabha Mukerjee, Rita Biswas, Kalyanmoy Deb, and Amrit P Mathur. Multi–objective evolutionary algorithms for the risk–return trade–off in bank loan management. *International Transactions in operational research*, 2002.

[32] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[33] Arvind Narayanan. Translation tutorial: 21 fairness definitions and their politics. In *Proc. Conf. Fairness Accountability Transp., New York, USA*, 2018.

[34] Eric WT Ngai, Yong Hu, Yiu Hing Wong, Yijun Chen, and Xin Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision support systems*, 50(3):559–569, 2011.

[35] Executive Office of the President. Big data: Seizing opportunities, preserving values, 2014.

[36] Executive Office of the President, Cecilia Munoz, Domestic Policy Council Director, Megan (US Chief Technology Officer Smith (Office of Science, Technology Policy)), DJ (Deputy Chief Technology Officer for Data Policy, Chief Data Scientist Patil (Office of Science, and Technology Policy)). *Big data: A report on algorithmic systems, opportunity, and civil rights*. Executive Office of the President, 2016.

[37] Patrick Oliver. Protesting the death of george floyd. 2020.

[38] Cathy O'neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Broadway Books, 2016.

[39] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. On fairness and calibration. In *Advances in Neural Information Processing Systems*, pages 5680–5689, 2017.

[40] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[41] Michael L Rich. Machine learning, automated suspicion algorithms, and the fourth amendment. *University of Pennsylvania Law Review*, pages 871–929, 2016.

[42] Sebastian Schelter, Yuxuan He, Jatin Khilnani, and Julia Stoyanovich. Fairprep: Promoting data to a first-class citizen in studies on fairness-enhancing interventions. *arXiv preprint arXiv:1911.12587*, 2019.

[43] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.

[44] Sahil Verma and Julia Rubin. Fairness definitions explained. In *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*, pages 1–7. IEEE, 2018.

[45] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 335–340, 2018.

Table 3: Comparison between models. mean ± standard deviation

		logistic regression	neural network	random forest
ACI	accuracy	0.852 ± 0.000	**0.855 ± 0.002**	0.844 ± 0.002
	roc_auc	0.904 ± 0.000	**0.908 ± 0.001**	0.889 ± 0.000
BM	accuracy	**0.901 ± 0.000**	**0.901 ± 0.002**	0.899 ± 0.001
	roc_auc	0.930 ± 0.000	**0.934 ± 0.001**	0.932 ± 0.001
COMPAS	accuracy	**0.677 ± 0.000**	0.641 ± 0.061	0.652 ± 0.006
	roc_auc	**0.725 ± 0.000**	0.679 ± 0.088	0.695 ± 0.002

A Additional Experiments and Details

In this section, we give additional details from the experiments in Section 6, as well as additional experiments.

The need for neural networks. We start by comparing the performance of neural networks to logistic regression and gradient-boosted regression trees (GBRT) on the datasets we used, to demonstrate the need for neural networks. This experiment is described at the start of Section 6. For convenience, we restate the details here. We construct a neural network with 10 fully-connected layers of size 32, BatchNorm for regularization, and a dropout rate of 0.2, and we compare this to logistic regression and GBRT on the ACI, BM, and COMPAS datasets. See Table 3. We see that the neural network achieves better accuracy and ROC AUC on all datasets except COMPAS, which is within one standard deviation of the optimal performance.

Bias sensitivity to initial model conditions. Next, we study the sensitivity of bias to initial model conditions. Recall that in Table 1, we computed the mean and standard deviation of three fairness measures, as well as accuracy, for training a neural network with respect to different initial random seeds. We see that standard deviation of the bias is an order of magnitude higher than the standard deviation of the accuracy. Now we run more experiments to show that the contribution of the weights to the bias score are sensitive to the initial random seed.

For this experiment, we train 10 neural networks with the same architecture as described in Section 6. We want to identify which parameters of the network contribute most to the bias. To identify these parameters, we create 1000 random delta vectors with mean 1 and standard deviation 0.1 for each of the neural networks. We then take the Hadamard product of each random delta vector with the parameters of the corresponding network. We then evaluate the statistical parity difference (SPD) on the test set for the networks with the new perturbed parameters. To identify which parameters contribute most to the bias, we train a linear model for each of the 10 neural networks to predict the bias from the random delta vectors, and then we analyze the coefficients of the corresponding linear models. The linear models are successfully able to predict the bias based on the random delta vectors with an R^2 score of 0.861 ± 0.090. Figure 2 (left) shows that only a small fraction of the parameters contribute to the majority of the bias.

Now we want to identify how similar the coefficients of the linear models are across all 10 neural networks. To identify this, we stack the normalized coefficients for the linear models and decomposed

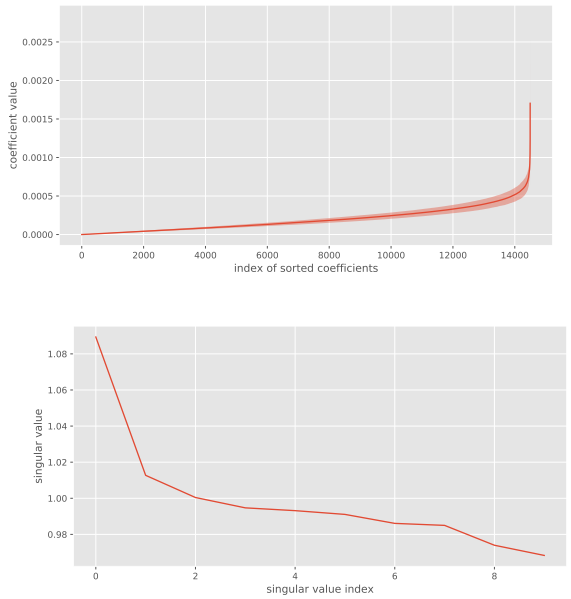

Figure 2: Results for coefficient analysis.

the stacked matrix with singular value decomposition. The singular values of the matrix measure the degree of linear independence between the coefficients for the 10 linear models. As we see from Figure 2 (right), the singular values are all close to 1. This indicates that the coefficients are all relatively different from one another. This means that the parameters of the 10 neural networks that correspond to the bias are different for each network indicating that each time we train a model, even if it has the same architecture, the parameters that contribute to bias are different.