2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT) | 978-1-6654-6090-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/BDCAT56447.2022.00041

A Predictor-Corrector Method for Multi-objective Optimization in Fair Machine Learning

Sean Wang
College of Arts & Sciences
Cornell University
Ithaca, NY, USA
sw665@cornell.edu

Arielle Carr
Computer Science & Engineering
Lehigh University
Bethlehem, PA, USA
arg318@lehigh.edu

Sihong Xie
Computer Science & Engineering
Lehigh University
Bethlehem, PA, USA
six316@lehigh.edu

***Abstract*—Issues of fairness often arise in graphical neural networks used for misinformation detection. However, improving fairness can often come at the cost of reducing accuracy and vice versa. Therefore, we formulate the task of balancing accuracy and fairness as a multi-objective optimization (MOO) problem where we seek to find a set of Pareto optimal solutions. Traditional first-order approaches to solving MOO problems such as multi-gradient descent can be costly, especially with large neural networks. Instead, we describe a more efficient approach using the predictor-corrector method. Given an initial Pareto optimal point, this approach predicts the direction of a neighboring solution and refines this prediction using a few steps of multi-gradient descent. We show experimentally that this approach allows for the generation of high-quality Pareto fronts faster than baseline optimization methods.**

***Index Terms*—multi-objective optimization, fair machine learning, iterative methods, neural networks**

I. Introduction

Misinformation on the Internet has become increasingly widespread. For instance, platforms such as Yelp and Amazon are frequently plagued by fake reviews. As a result, machine learning algorithms have been adopted for the purpose of detecting reviews that have a high likelihood of being spam. However, the quality of such algorithms cannot be measured using accuracy as a sole metric. Since these algorithms cannot feasibly reach 100% accuracy, certain biases arise that can harm some users. For example, users who have posted fewer reviews are more likely to have their posts falsely flagged as misinformation. Thus, we seek to minimize such unfairness while still maintaining a sufficient level of accuracy. Our goal is to optimize these two objectives: We want to reduce detection bias without drastically affecting the detection accuracy.

One challenge with optimizing multiple objectives is that they can often conflict with each other. That is, reducing unfairness can come at the cost of accuracy and vice versa. Therefore, it may not always be the case that there exists a single solution for which both unfairness and inaccuracy are minimized. Past approaches have tried to circumvent this issue by defining objective weights based on relative importance of individual objectives so that a single optimal point can be found. However, this process is often time-consuming and problem specific [4]. Furthermore, users may want to compare different trade-offs according to different objective weights such as when a user is willing to sacrifice accuracy in exchange for greater fairness. In this case, a single solution is not always sufficient.

We consider multi-objective optimization (MOO) methods that allow us to explore different trade-offs by generating a set of solutions known as the Pareto front [4]. Traditional first-order MOO methods, such as multi-gradient descent (MGD) [11], are computationally expensive on large graphs due to the gradient computation at each iteration, and requiring many iterations to generate a Pareto front.

We propose a more efficient approach using the predictor-corrector method introduced in [1]. At a given Pareto optimal point, the direction to a neighboring optimal point can be approximated by the solution to a linear system representing the tangent plane at the original point. This system can be solved efficiently using iterative methods such as MINRES or Conjugate Gradient. Then, this initial approximation is refined using stochastic gradient descent. We demonstrate experimentally that these methods result in reduced computation time with minimal sacrifice in Pareto front quality.

II. Preliminaries

A. A Graph Neural Network for Spam Detection

In order to predict which reviews have the highest likelihood of being fake, we use a graph neural network (GNN) containing layers with nodes representing users, products, and reviews. The objectives we wish to minimize are the losses of this network with respect to accuracy and fairness.

We measure accuracy loss using the normalized discounted cumulative gain (NDCG) metric:

$$\frac{1}{Z}\sum_{j=1}^{n}\mathbb{1}[y_j=1]\frac{1}{\log\left(r_j+1\right)}. \tag{1}$$

Here, r_j is the rank of the jth labeled review according to its probability of being fake as predicted by the GNN model, and Z is the maximum possible NDCG score used for normalization.

We measure fairness loss using cross-NDCG (xNDCG), which compares the equality in accuracy across two groups, the favored group and the protected group. The favored group (denoted $A = 0$) contains the top 30% of users based on

978-1-6654-6090-3/22/$31.00 ©2022 IEEE
DOI 10.1109/BDCAT56447.2022.00041

review count, and the protected group (denoted $A = 1$) contains the remaining users in our dataset. xNDCG measures the similarity in the NDCG scores of these groups as

$$\left| \frac{1}{Z_0} \sum_{j=1}^{n_0} \mathbb{1}[y_j = 1, A_j = 0] \frac{1}{\log(r_j^0 + 1)} - \frac{1}{Z_1} \sum_{j=1}^{n_1} \mathbb{1}[y_j = 1, A_j = 1] \frac{1}{\log(r_j^1 + 1)} \right|. \quad (2)$$

We can describe an MOO problem for optimizing these losses by a vector-valued function $f(x)$:$\mathbb{R}^n \to \mathbb{R}^m$ where x is a vector containing the parameters for our neural network and m is the number of objective functions we want to optimize. That is, each component function $f_i : \mathbb{R}^n \to \mathbb{R}$ represents one of our loss functions to be optimized.

B. Predictor Corrector Method

Given a point of a function, the predictor-corrector method allows us to approximate the value of that function at a nearby point. This method consists of two steps (see Fig. 1). First, in the predictor step, we determine an approximate direction to a neighboring point, and then move along that direction based on a predetermined, fixed step size. Then, we refine this initial approximation in the corrector step. We describe each of these steps in further detail below.

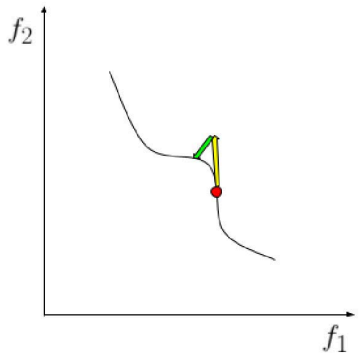

Fig. 1: The predictor step (yellow) and corrector step (green) given an initial point (red).

1) Predictor Step: In order to determine the direction in which to move, we solve for the vector v in the following linear system derived in [3]:

$$H(x_0^*)v = \nabla f(x_0^*)\beta \quad (3)$$

where x_0^* is some starting Pareto optimal point, $H(x_0^*)$ and $\nabla f(x_0^*)$ are respectively the Hessian and Jacobian of our loss functions at x_0^*, and β is a weighting vector $\begin{pmatrix} a & b \end{pmatrix}$ where $a, b \in [0, 1]$. Here, β determines which direction along the Pareto front to move in relation to x_0^*. Intuitively, this approach works by finding a suitable direction to move along the tangent plane of f at x_0^* as depicted in Fig. 1.

Solving this linear system directly is impractical as computing and inverting the Hessian is prohibitively expensive, requiring $O(n^3)$ time. Furthermore, for our purposes, the Hessian matrices are both large and dense, meaning that the storage of full Hessian matrices is infeasible. Instead, we use iterative solvers, which allow us to efficiently solve the system without the need for explicit computation of the Hessian. In particular, we consider the minimal residual method (MINRES) and the Conjugate Gradient method (CG) discussed in detail in the next section. MINRES and CG require only a linear operator to compute a matrix-vector product on the left-hand side of Eq. (3).

2) Corrector Step: For our experiments, we use a single step of multi-gradient descent from [11] as the corrector step. Any other multi-objective optimization method could be used as well. The choice in method for this step is made to save running time with minimal effect on Pareto front quality.

III. Iterative Methods

We start by describing basic first-order and second-order methods to motivate the CG and MINRES methods.

A. Gradient Descent

Gradient Descent (GD) is a basic first-order optimization method for convex functions. Given an initial guess x_0 for the minimum of a function $f(x)$, GD iteratively updates x_0 using the direction of steepest descent. Since the gradient of f is the direction of maximal increase, the direction of steepest descent is simply the direction opposite to the gradient of f at x_0. Then, x_0 is updated by moving a predetermined fixed step size parameter (learning rate) α along the computed direction. This process is repeated until a specified maximum number of iterations, $maxiter$, is reached, or until the magnitude of the update is less than a certain tolerance, tol. Algorithm 1 provides an outline of the GD method.

Algorithm 1 Gradient Descent Algorithm

```
i = 0
while i < maxiter do
    x_i = x_0
    d = ∇f(x_i)
    if αd < tol then
        break
    end if
    x_i = x_i − αd
end while
```

There also exist variations of GD such as multiple gradient descent (MGD), which optimizes multiple objectives, and stochastic gradient descent (SGD), which introduces random perturbations at each iteration. However, there are two main issues that come with using GD and its variations. First, the convergence rate of GD can vary depending on the choice of α. For instance, GD can take much longer to converge in low curvature scenarios for small α [5]. Additionally, when using GD, there exists a possibility of stopping at a local minimum

rather than the global minimum of the function. Both of these issues arise from the fact that GD is a first-order method and no second-order curvature information is used.

B. Newton's Method

Newton's method (NM) is a second-order optimization method that seeks to address the issues with GD by utilizing information about the second-order derivative of f. NM is very similar to GD, with the main difference being that NM scales the descent direction based on the curvature of f at the given point. That is, at each iteration of NM,

$$x_{i+1} = x_i - H(x_i)^{-1}\nabla f(x_i)$$

where H is the Hessian of f at x_i. However, this requires explicit computation and inversion of the Hessian, which is far too costly for our method. An iterative linear solver avoids computation of the Hessian matrix.

C. CG and MINRES

Conjugate Gradient (CG) [10] and MINRES [8] are iterative methods for approximately solving a linear system $Ax = b$ where A is symmetric. Both methods belong to a class of linear solvers known as matrix-free solvers, meaning that the storage of the matrix A is not required. Instead, we only need to define the matrix-vector product Av for an arbitrary vector v using a linear operator. Both of these methods are known to converge monotonically, so we are able to improve runtime using early termination. Therefore, a restriction on the maximum number of iterations is imposed on these methods to ensure a reasonable runtime.

1) Conjugate Gradient: The conjugate gradient method solves the linear system $Ax = b$ as a more efficient version of the gradient descent algorithm. CG typically converges in much fewer iterations and addresses the issues with step size present in GD. CG operates by repeatedly finding optimal points along conjugate directions. That is, at iteration k, given a point x_k and a direction p_k, CG updates x_k as

$$x_{k+1} = x_k + \alpha p_k, \tag{4}$$

where

$$\alpha = \frac{p_k^T(Ax_k + b)}{p_k^T A p_k}. \tag{5}$$

Note that in (5), we require the computation of a matrix-vector product Ax. We will describe later how this can be done without the explicit storage of A. Then, a new direction conjugate to p_k is computed. Intuitively, the requirement that directions be conjugate is enforced to ensure that minimization along the current iteration has no effect on work done in previous iterations. In other words, at the kth iteration we minimize f along a certain direction p_k. Requiring p_{k+1} to be conjugate to p_k ensures that as we move in the direction p_{k+1}, our approximate solution is still minimized along the direction p_k.

This method is guaranteed to converge in at most n iterations, where n is the dimension of A, but CG often reaches an acceptable tolerance in much fewer iterations. However, it is important to note that CG requires the matrix A to be positive-definite [9]. Since our Hessian matrices are not guaranteed to be positive-definite, CG may not always work.

2) MINRES: The MINRES method solves the system $Ax = b$ by minimizing the residual at the kth iteration. The Krylov subspace of dimension k is defined as

$$\mathcal{K}^k(A, r_0) = span\{r_0, Ar_0, ..., A^{k-1}r_0\}, \tag{6}$$

and is augmented incrementally with each iteration of MINRES. Note that $\mathcal{K}$ is never explicitly computed.

The residual of an approximate solution x_k is a measure of how well x_k approximates the exact solution and is defined as

$$r_k = Ax_k - b. \tag{7}$$

Then, at iteration k of MINRES, we find

$$z_k \in \mathcal{K}^k(A, r_0)$$

and compute $x_k = x_0 + z_k$ such that the residual norm

$$||r_k||_2 = ||Ax_k - b||_2 \tag{8}$$

is minimized. This continues until (8) is within a certain tolerance, which is typically specified by the user. Also, note that unlike CG, MINRES works for indefinite matrices [8]. Therefore, we choose to use MINRES in our experiments as our matrices are not always positive definite.

D. Hessian-Vector Products

Recall that for the predictor step, we want to solve the linear system $H(x_0^*)v = \nabla(x_0^*)\beta$ without using direct methods. When using matrix-free methods such as CG and MINRES, we are able to solve the given linear system without explicitly storing H by instead finding the product of the Hessian with an arbitrary vector v. In fact, this computation constitutes a large portion of the computational cost at each iteration of CG and MINRES. Therefore, a primary concern is not only to define this product to avoid storage of H, but also to ensure the efficient computation of Hessian-vector products (HVPs). One method for doing so is the finite difference method:

$$Hv = \lim_{\epsilon \to 0} \frac{\nabla f(x + \epsilon v) - \nabla f(x)}{\epsilon}.$$

By choosing ϵ to be some positive value close to 0, we can closely approximate the value of the product Hv using only two gradient computations [5]. This can be done using two instances of automatic differentiation (AD). However, this method is not numerically stable, so we use another method known as the Pearlmutter algorithm [5]. This algorithm allows for computation of Hv for neural networks in $O(n)$ time by using both forward and backwards passes through the neural network. In our approach, this is done using two backpropagation calls.

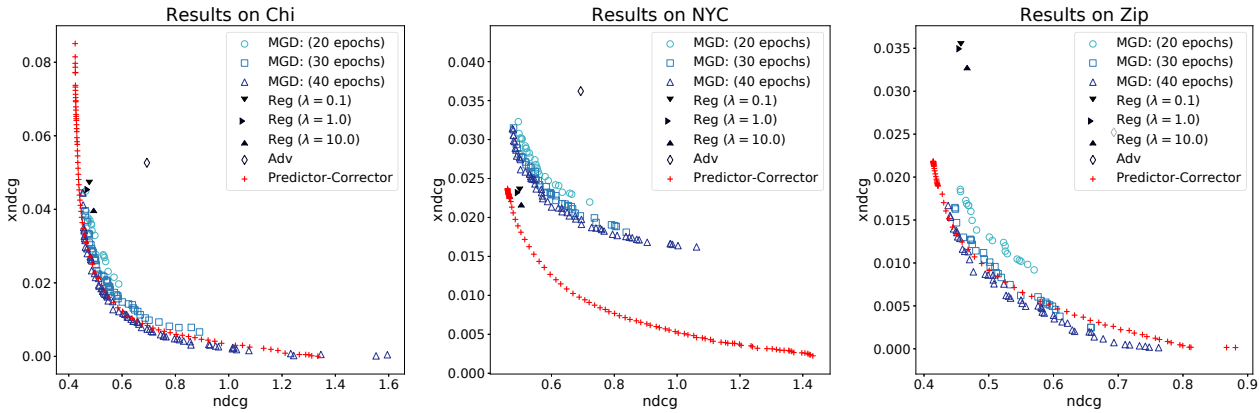

Fig. 2: Pareto front comparison on YelpChi, YelpNYC, and YelpZip datasets

IV. Experiments

A. Experimental Setup

We compare the performance of our algorithm using predictor-corrector method (PC) and MGD on three spam-detection datasets: YelpChi, YelpNYC, and YelpZip based on results given in [1]. The sizes of these datasets and their corresponding graph neural network sizes are given in Table I. For these experiments, we choose the favored group to contain users in the top 30% based on review count and the protected group to contain the remaining users. As described in Section II, we use NDCG and xNDCG to measure accuracy and loss, respectively.

These experiments were run with a graph neural network, with an initial Pareto optimal solution generated using 75 optimization steps of MGD. The baseline MGD was tested at 20, 30, and 40 training epochs, which were optimized by solving quadratic programming problems. A learning rate of 0.005 was found to offer the best performance. 1. The predictor step used 50 maximum iterations of MINRES, based on ablation studies in [4]. The predictor and corrector step used a step size 0.1 and 0.01 respectively, as these values were found to minimize runtime while maintaining Pareto front quality.

TABLE I: Dataset and model sizes

Datasets	Products	Reviews	Users	Model size
YelpChi	201	67395	38063	1234
YelpNYC	923	358911	160220	1234
YelpZip	5044	608598	260277	1234

1) Pareto Front Quality: We perform a visual comparison of the Pareto fronts generated on each of our methods. As shown in Fig. 2, on YelpNYC, the Pareto front generated by PC noticeably outperforms those of MGD for all values of epochs. However, on YelpChi and YelpZip, we see that the PC front is only slightly better than the fronts of MGD-20 and MGD-30, and is slightly worse than the front generated by MGD-40. Also note that on all three datasets, PC generates a much larger Pareto front than all three MGD variants, resulting in more expansive Pareto fronts that span a wider range of the loss values.

B. Runtime Comparison

Fig. 3 shows a runtime comparison between the different algorithms for Pareto front generation. We see that on YelpNYC, PC clearly outperforms MGD since for all MGD variants, PC generates a higher quality Pareto front with a faster runtime. The only case where MGD outperforms PC is MGD-20 on YelpChi and YelpZip. However, in this case, the MGD-20 front is worse than the PC front. Seeing as the difference in runtime is slight, we cannot say that MGD-20 outperforms PC, especially when Pareto front quality is taken into account. When MGD-40 produces higher quality Pareto fronts, we see that the runtime is significantly higher than that of MGD. In the case of YelpZip, PC runs over three times faster than MGD-40. Since the fronts generated by MGD-40 are only slightly better than the PC fronts, the small decrease in Pareto front quality is an acceptable trade-off for the significant reduction in runtime.

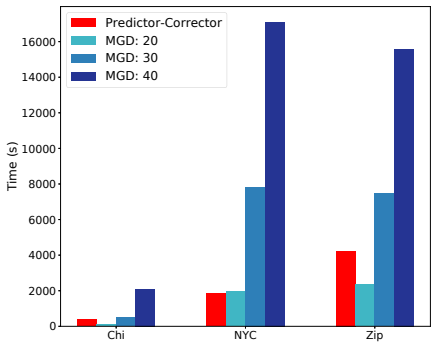

Fig. 3: Comparison of running time of different algorithms on the datasets.

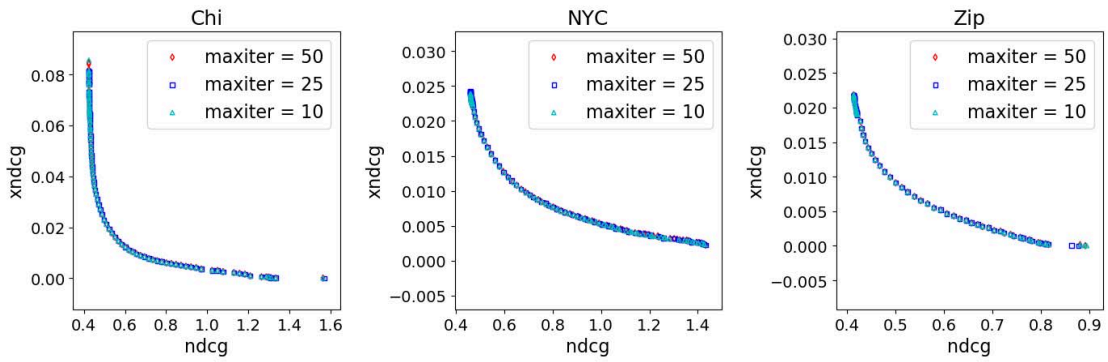

Fig. 4: Pareto fronts generated by the predictor-corrector method for varying values of maximum MINRES iterations

C. Effect of the Maximum Number of Iterations

We also measure the effect that decreasing the maximum number of iterations for the linear solver has on Pareto front quality and runtime. We test two new values for *maxiter*, 25 and 10, and compare the results to our previous experiments that used *maxiter* = 50.

Fig. 4 demonstrates how Pareto front quality is affected by decreasing the maximum number of iterations for MINRES. We see that as $maxiter$ decreases, there is almost no effect on Pareto front quality. On all three datasets, the fronts generated using 10 and 25 MINRES iterations are nearly identical to each other and to the baseline front generated with $maxiter = 50$.

Looking at the runtime comparison depicted in Fig. 5, we see that when $maxiter = 25$ or $maxiter = 10$, there is a noticeable decrease in runtime from our original method. Thus, by reducing the number of iterations before MINRES terminates, we can improve the runtime of our algorithm with minimal impact on the quality of the Pareto front.

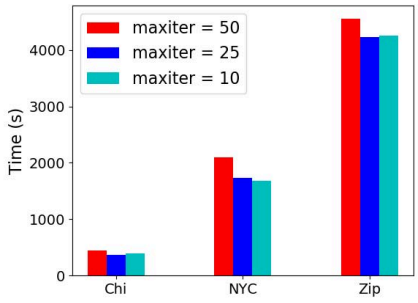

Fig. 5: Runtime comparison for varying maximum MINRES iterations

D. Discussion

From our experimental results, we see that the predictor-corrector method outperforms or is comparable with MGD on all three datasets (YelpChi, YelpNYC, YelpZip), both in Pareto front quality and runtime. On YelpNYC, PC generates a Pareto front better than all three MGD fronts with a significantly lower runtime. On YelpChi and YelpZip, the Pareto front generated PC is better than all of the MGD fronts except when the number of training epochs is 40. However, the PC front in these cases includes a wider range of different tradeoffs and is only slightly worse than the MGD front. Furthermore, we see that PC runs significantly faster in these scenarios, so PC still outperforms MGD.

We also show experimentally that the performance of the PC algorithm can be further improved by decreasing the number of maximum iterations allowed for the linear solver. Reducing the maximum number of iterations leads to decreased runtime while the resulting Pareto fronts remain largely unchanged. From this we see that the linear solver plays an important role in the efficiency of the PC algorithm, suggesting that further improving the efficiency of these solvers will improve the ability to quickly and accurately compute Pareto fronts.

V. Conclusion and Future Work

We demonstrate the efficiency of a predictor-corrector method for generating Pareto optimal networks for fair spam-detection. We see that in many cases, the predictor-corrector method outperforms traditional multi-objective optimization methods, both in terms of Pareto front quality and runtime. Furthermore, in cases where traditional methods yield more accurate fronts, the predictor-corrector method runs significantly faster with only a slight reduction in quality. Hence, we see the the benefit of using this method for solving multi-objective optimization problems.

With regards to future improvements to the method, we can further reduce the time needed for Hessian-vector product computations in our linear solvers. One option is to explore the use of the Gauss-Newton approximation to the Hessian [2] [5], which has the form

$$H \approx J^T J$$

where J is the Jacobian of our loss functions. One advantage that this approximation has over the Hessian is that it is guaranteed to be positive semi-definite, allowing us the flexibility to use CG if desired [2]. Furthermore, there exists a

generalized version of Pearlmutter's algorithm for computing the matrix-vector product with the Gauss-Newton matrix. In fact, this algorithm runs faster than Pearlmutter's algorithm and uses less memory [5] [7]. However, this method's reliance on forward-mode AD, combined with the current lack of support for forward AD in the Pytorch deep-learning framework, means that another method for finding the matrix-vector product may be needed.

ACKNOWLEDGEMENT

Sihong Xie is supported in part by the National Science Foundation under Grants NSF IIS-1909879, NSF CNS-1931042, NSF IIS-2008155, and NSF IIS-2145922. Sean Wang is supported by the National Science Foundation under Grants NSF CNS-2051037. Any opinions, findings, conclusions, or recommendations expressed in this document are those of the author(s) and should not be interpreted as the views of the National Science Foundation.

REFERENCES

[1] Enouen, Eric & Mathesius, Katja & Wang, Sean & Carr, Arielle & Xie, Sihong. (2022). Efficient first-order predictor-corrector multiple objective optimization for fair misinformation detection. 10.48550/arXiv.2209.07245.

[2] Gargiani, M., Zanelli, A., Diehl, M., & Hutter, F. (2020). On the promise of the stochastic generalized Gauss-Newton method for training DNNs. arXiv preprint arXiv:2006.02409.

[3] Hillermeier, C. (2001). Nonlinear multiobjective optimization: a generalized homotopy approach (Vol. 135). Springer Science & Business Media.

[4] Ma, P., Du, T., & Matusik, W. (2020, November). Efficient continuous pareto exploration in multi-task learning. In International Conference on Machine Learning (pp. 6522-6531). PMLR.

[5] Martens, J. (2010, June). Deep learning via hessian-free optimization. In ICML (Vol. 27, pp. 735-742).

[6] Pearlmutter, B. A. (1994). Fast exact multiplication by the Hessian. Neural computation, 6(1), 147-160.

[7] Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. Neural computation, 14(7), 1723-1738.

[8] Black, N., & Moore, S. (n.d.). Minimal residual method (E. Weisstein, Ed.). Mathworld.wolfram.com. From https://mathworld.wolfram.com/MinimalResidualMethod.html

[9] Yousef Saad. Iterative Methods for Sparse Linear Systems, 2nd Ed. SIAM, 2003.

[10] Hestenes, M.R., & Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. Journal of research of the National Bureau of Standards, 49, 409-435.

[11] Liu, S., & Vicente, L. N. (2019). The stochastic multi-gradient algorithm for multi-objective optimization and its application to supervised machine learning.