Journal of Computational and Graphical Statistics

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/ucgs20

Generative Multi-purpose Sampler for Weighted M-estimation

Minsuk Shin, Shijie Wang & Jun S. Liu

To cite this article: Minsuk Shin, Shijie Wang & Jun S. Liu (08 Dec 2023): Generative Multi-purpose Sampler for Weighted M-estimation, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2023.2292668

To link to this article: https://doi.org/10.1080/10618600.2023.2292668

Generative Multi-purpose Sampler for Weighted M-estimation

Minsuk Shin[a], Shijie Wang[a,*], Jun S. Liu[b]

[a]Department of Statistics, University of South Carolina,

[b]Department of Statistics, Harvard University December 8, 2023

*SHIJIEW@email.sc.edu

Abstract

To overcome computational bottlenecks of various data perturbation procedures such as the bootstrap and cross validations, we propose the *Generative Multi-purpose Sampler* (GMS), which directly constructs a generator function to produce solutions of weighted M-estimators from a set of given weights and tuning parameters. The GMS is implemented by a single optimization procedure without having to repeatedly evaluate the minimizers of weighted losses, and is thus capable of significantly reducing the computational time. We demonstrate that the GMS framework enables the implementation of various statistical procedures that would be unfeasible in a conventional framework, such as iterated bootstrap procedures and cross-validation for penalized likelihood. To construct a computationally efficient generator function, we also propose a novel form of neural network called the *weight multiplicative multilayer perceptron* to achieve fast convergence. An `R` package called `GMS` is provided, which runs under `Pytorch` to implement the proposed methods and allows the user to provide a customized loss function to tailor to their own models of interest.

Keywords: Weighted M-estimation, Bootstrap/resampling, Cross-validation, Scalable Computation, Iterated Bootstrap

1 Introduction

Consider a canonical setting in which $\mathbf{y} = \{y_1, \ldots, y_n\}$ are i.i.d. observations following a statistical model with the parameter of interest denoted by $\theta \in \Theta \subset \mathbb{R}^p$ ($n > p$). In some instances such as regression analysis, one may also include predictors or covariate variables for each observation. An efficient estimator of θ can often be found by solving the following (penalized) optimization problem: $\hat{\theta} = \operatorname{argmin}_\theta L_{\mathbf{y}}(\theta)$, where $L_{\mathbf{y}}(\theta) \equiv \frac{1}{n}\sum_{i=1}^{n} \ell_\eta(\theta; y_i)$ with $\ell_\eta(\cdot)$ being a suitable loss function with an auxiliary parameter η. The resulting $\hat{\theta}$ is often referred to as an *M-estimator* (Huber, 1992). For example, the maximum likelihood estimator (MLE) is a special M-estimator with the loss function being set as the negative log-likelihood function.

To assess the variability of the M-estimator $\hat{\theta}$, we study behaviors of the following tunable weighted M-estimators as inspired by the bootstrap methods (Efron, 1979):

$$\hat{\theta}_{\mathbf{w},\lambda,\eta} = \operatorname*{argmin}_\theta \left[\frac{1}{n}\sum_{i=1}^{n} w_i \ell_\eta(\theta; y_i) + \lambda u(\theta) \right] \overset{\Delta}{=} \operatorname*{argmin}_\theta L_{\mathbf{y}}(\theta; \mathbf{w}, \lambda, \eta), \quad (1)$$

where $\eta \in \mathbb{R}^+$ is an auxiliary parameter of the loss, $u(\cdot)$ is a penalty function on the parameter with a tuning parameter λ that can be set to zero for non-penalized settings, and $\mathbf{w} = (w_1, \ldots, w_n)^\top \in \mathcal{W}$ is a vector of weights following distribution $\pi(\mathbf{w})$. The auxiliary parameter η tunes the loss function. For example, in quantile regression models, $\eta \in (0,1)$ represents the quantile level and the loss function takes the form $\ell_\eta(\theta; y_i, X_i) = \rho_\eta(y_i - X_i^\top \theta)$, where $\rho_\eta(t) = t(\eta - I(t < 0))$. When the loss function has no auxiliary parameter, we simply denote the loss and the resulting estimator by $\ell(\theta; y_i)$ and $\hat{\theta}_{\mathbf{w},\lambda}$, respectively.

The formulation of (1) applies to a wide range of statistical procedures. For example, the classical bootstrap procedure of Efron (1979) corresponds to $\mathbf{w} \sim \text{Multinom}(n, 1_n / n)$, where 1_n is a n-dimensional vector of one, and $u(\theta) = 0$. Random-weight bootstrap procedures can be formulated by imposing a general

distribution on $\mathbf{w}$ that has a mean of one, finite variance, and sum to n. Its theoretical properties such as consistency have been studied (Præstgaard and Wellner, 1993; Cheng and Huang, 2010; Barbe and Bertail, 2012). A special and most well-known form of the random-weight bootstrap is to set $\mathbf{w} \sim n \times \mathrm{Dirichlet}(n; 1_n)$ as in the *Bayesian Bootstrap* (Rubin, 1981) and *Weighted Likelihood Bootstrap* (Newton and Raftery, 1994). Theoretical investigations and improvements of the bootstrap methods have been considered in a large body of literature (Chatterjee et al., 2005; McCarthy et al., 2018; Hall and Martin, 1988; Efron, 1987; Hahn, 1995; Kleiner et al., 2014).

Iterated bootstrap procedures are often employed to reduce the bias associated with a statistical inference procedure and/or improve the coverage precision of confidence intervals (Hall and Martin, 1988). A most frequently cited procedure is the double bootstrap, which first bootstraps and infers the parameter or prediction, and then estimates the bias of each bootstrapped solution via a second-level bootstrap. In (1), the double bootstrap procedures can be represented by setting a hierarchical weight distribution such that $\mathbf{s} = \{s_1, \ldots, s_n\} \sim \mathrm{Multinom}(n, 1_n / n)$ and $\mathbf{w} \mid \mathbf{s} \sim \mathrm{Multinom}(n, \mathbf{s} / n)$. These iterated bootstrap methods can be shown to provide more accurate confidence coverage (i.e., the second or higher-order accuracy) compared with single bootstraps and asymptotic approximations Martin, 1992; McCarthy et al., 2018; Hall, 2013; Lee and Young, 1999, 1995. However, iterative bootstraps are computationally very expensive and are rarely used in practice when the data are of moderate to large sizes.

The tunable weighted M-estimation in (1) can also represent *K*-fold cross-validation. For pre-selected folds, such as a group of sample indices $I_1, \ldots, I_K$, we set $w_i = 0$ for i in the fold of interest, say I_1, and set $w_i = 1$ in all other folds. This means that the observations in I_1 will be ignored during training, rendering I_1 to be test samples. If $u(\cdot) = \|\cdot\|_1$, the evaluated $\hat{\theta}_{\mathbf{w},\lambda}$ is equivalent to the LASSO estimator (Tibshirani, 1996), based on a tuning parameter λ, trained without using the samples in the considered fold I_1, resulting in a cross-validated LASSO. The computational burden of the cross-validation linearly increases with the fold size K and the candidate set size of the

tuning parameter, and a typical amount is at least a few hundreds of repetitive computations.

While aforementioned weighted M-estimation procedures are widely used in statistics and science, the computational bottleneck caused by their repetitive nature poses significant practical difficulties. To alleviate these computational difficulties, we propose a computational approximation strategy based on a neural network-based generative process, called the *Generative Multi-purpose Sampler* (GMS) (with the *Generative Bootstrap Sampler* (GBS) as a special case for bootstrap). Instead of repeating the same optimization process for various combinations of weights $\mathbf{w}$'s and parameters λ's and η's, the GMS constructs a generator function that takes $(\mathbf{w}, \lambda, \eta)$ as input and approximates the corresponding weighted M-estimator $\hat{\theta}_{\mathbf{w},\lambda,\eta}$. In addition to taking advantage of the high representation power of neural networks, a key idea for the GMS to achieve the desired computational efficiency gain is to minimize an integrative loss in its training, which optimizes both the M-estimation and the parameters employed by the GMS simultaneously.

The rest of the article is organized as follows. Section 2 introduces the general GMS framework and uses a toy example to explain its potential gains. Section 3 details its specialization for the bootstrap, namely the *Generative Bootstrap Sampler* (GBS). Section 4 discusses the training of GMS for cross-validation with Lasso and quantile regression. Section 6 provides details on the neural network structures and detailed computational aspects of GMS. Section 7 concludes with a brief discussion.

2 Generative Multi-purpose Sampler

2.1 The basic formulation

We view the weighted M-estimator $\hat{\theta}_{\mathbf{w},\lambda,\eta}$ as a function of the weight $\mathbf{w}$, the tuning parameter λ, and the auxiliary parameter η, i.e., $G(\mathbf{w}, \lambda, \eta)$, and attempt to approximate it by a member in a suitable family of functions $\mathcal{G} = \{G_\phi : \mathbb{R}^{n+2} \mapsto \mathbb{R}^p, \phi \in \Phi\}$, where Φ is the space of parameters that characterize a function in the family. By doing so, we turn the unrestricted optimization problem in

(1) into a restricted optimization problem in a functional space, i.e., finding a proper parameter of the generator function such that, for all $\mathbf{w} \in \mathcal{W}, \lambda \in \mathbb{R}^+$, and $\eta \in \mathbb{R}^+$,

$$\hat{\phi} = \underset{\phi \in \Phi}{\operatorname{argmin}} L_{\mathbf{y}}(G_\phi(\mathbf{w}, \lambda, \eta); \mathbf{w}, \lambda, \eta), \quad (2)$$

A slightly less ambitious, but more robust, formulation is to solve

$$\hat{\phi} = \underset{\phi \in \Phi}{\operatorname{argmin}} \mathbb{E}_{\mathbf{w},\lambda,\eta}\left[L_{\mathbf{y}}(G_\phi(\mathbf{w}, \lambda, \eta); \mathbf{w}, \lambda, \eta)\right], \quad (3)$$

where $\mathbb{E}_{\mathbf{w},\lambda,\eta}(\cdot)$ is taken with respect to a proper distribution of $(\mathbf{w}, \lambda, \eta)$ defined on $\mathcal{W} \times \mathbb{R}^+ \times \mathbb{R}^+$. We name this generative framework in (3) as the GMS. For non-penalized settings without the auxiliary parameter η, we simply denote the generator function by $G(\mathbf{w})$. We also use the notation $\hat{G} = G_{\hat{\phi}}$. The weight distribution for Efron' s nonparametric bootstrap is simply $\mathbf{w} \sim \text{Multinom}(n, 1_n / n)$. For the Bayesian bootstrap (Rubin, 1981), $\mathbf{w} / n \sim \text{Dirichlet}(n, 1_n)$. The distributions of λ and η can simply be the uniform distribution on candidate sets of λ's and η's chosen by the researcher. Another reasonable distribution of λ and η is to add random noises to a discrete set of candidate values (see Section 6.3 for details).

Suppose that $\hat{\phi}$ is the solution of (3) for a sufficiently large family $\mathcal{G}$ and a proper distribution on $\{\mathbf{w}, \lambda, \eta\}, \mathbb{P}_{\mathbf{w},\lambda,\eta}$, supported on $\mathcal{W} \times \mathbb{R}^+ \times \mathbb{R}^+$. If the solution $\hat{\theta}_{\mathbf{w},\lambda,\eta}$ of (1) is unique for any given $(\mathbf{w}, \lambda, \eta)$ in the support, then $G_{\hat{\phi}}(\mathbf{w}, \lambda, \eta)$ should be very close to $\hat{\theta}_{\mathbf{w},\lambda,\eta}$ almost surely in $\mathbb{P}_{\mathbf{w},\lambda,\eta}$. It is easy to see this point by contradiction – if not, then there exist $\epsilon > 0$ and a subset $S^* \subset \mathcal{W} \times \mathbb{R}^+ \times \mathbb{R}^+$ such that $\mathbb{P}_{\mathbf{w},\lambda,\eta}(S^*) > 0$ and $G_{\hat{\phi}}(\mathbf{w}, \lambda, \eta) \leq \hat{\theta}_{\mathbf{w},\lambda,\eta} - \epsilon$ on S^*. Thus, we can find another function that differs from $G_{\hat{\phi}}$ only on S^* and achieves a smaller value in (3).

A main takeaway from this argument is that optimizing the integrative loss over the space of $(\mathbf{w}, \lambda, \eta)$ instead of the individual loss is appropriate for training. To benefit from this formulation, we must choose an appropriate family $\mathcal{G}$ of functions G_ϕ and a suitable distribution $\mathbb{P}_{\mathbf{w},\lambda,\eta}$ to cover the hyperparameter space of interest. As

demonstrated by our empirical studies on a wide range of problems, restricting $\mathcal{G}$ to be a class of neural networks and choosing a reasonable distribution $\mathbb{P}_{\mathbf{w},\lambda,\eta}$ appears to work well (see details in Section 6.3).

As shown in Cybenko (1989) and Lu et al. (2017), *Multi-Layer Perceptrons* (MLP), or equivalently, *Feed-forward Neural Networks* (FNNs), are theoretically capable of approximating any Lebesgue integrable function when the numbers of neurons and layers are sufficiently large. Also, recent successful applications of deep neural networks in a variety of data-rich fields provide compelling evidence supporting the use of over-parameterized MLPs and other types of neural networks for approximating extremely complicated functions (Goodfellow et al., 2014; Arjovsky et al., 2017). To train a neural network to achieve the task in (3), we employ a backpropagation algorithm (Rumelhart et al., 1986) along with *Stochastic Gradient Descent* (SGD) and its variants. More details are given in Section 6.1.

2.2 Intuitions for potential gains

Imagine that we have independent weight vectors $(\mathbf{w}^{(1)},\lambda^{(1)},\eta^{(1)}),\ldots,(\mathbf{w}^{(M)},\lambda^{(M)},\eta^{(M)})$ from $\mathbb{P}_{\mathbf{w},\lambda,\eta}$, we can approximate the expectation in (3) by

$$\mathbb{E}_{\mathbf{w},\lambda,\eta}\left[L_{\mathrm{y}}(G(\mathbf{w},\lambda,\eta);\mathbf{w},\lambda,\eta)\right]\approx\frac{1}{M}\sum_{m=1}^{M}L_{\mathrm{y}}(G(\mathbf{w}^{(m)},\lambda^{(m)});\mathbf{w}^{(m)},\lambda^{(m)},\eta^{(m)}). \quad (4)$$

M do not need to be very large (*M*=100, say) since a small number of samples of $(\mathbf{w},\lambda,\eta)$ can be generated continuously within the iterative SGD algorithm to aid the fitting: after updating the FNN parameter ϕ with SGD based on (4), we use the newly created samples to evaluate the fit and to provide refreshed gradient. Thus, the two optimization tasks, i.e., minimizing the loss function L_{y} and finding optimal ϕ for the generator $G(\cdot)$, co-evolve and help each other.

If we were to cast the task of training a generator in a classical machine learning framework, we would have to first obtain a set of training samples, $\{(\mathbf{w}^{(b)},\lambda^{(b)},\hat{\theta}^{(b)})\}_{b=1}^{B}$, where $\hat{\theta}^{(b)}=\hat{\theta}_{\mathbf{w}^{(b)},\lambda^{(b)}}$, by evaluating *B* optimizations in (1) with $(\mathbf{w}^{(b)},\lambda^{(b)})$ for

$b = 1, \ldots, B$ (ignoring η for simplicity in this case). Then, one may try to learn a function g by minimizing

$$\hat{g} = \underset{g}{\operatorname{argmin}} \sum_{b=1}^{B} \left\| \hat{\theta}^{(b)} - g(\mathbf{w}^{(b)}, \lambda^{(b)}) \right\|^2, \quad (5)$$

under the l_2-distance $\|\cdot\|$. However, this squared-loss only measures the distance between the fitted generator $\hat{g}(\mathbf{w}, \lambda)$ and its training true value $\hat{\theta}_{\mathbf{w},\lambda}$. As a result, it cannot inform us how to improve the fitting of the original statistical loss in (1) other than a simple interpolation. Thus, the function trained in this manner tends to be inaccurate if B is small, or may be prohibitively expensive in computation if we must rely on a large B, in which case computational advantages of the generative process would be non-existing or limited.

Training the generator function G in conjunction with minimizing the loss function via the GMS formulation (3) is significantly more efficient. The classical loss (5) fits only on the training data with a limited size, $\{(\mathbf{w}^{(b)}, \lambda^{(b)}, \hat{\theta}^{(b)})\}_{b=1}^{B}$, resulting in an over-fitting issue. The GMS, on the other hand, is trained using the weights and tuning parameters generated from a predefined distribution without requiring additional optimizations for (1), and generating $\mathbf{w}$ and λ is nearly cost-less. As a result, the GMS training procedure not only seeks the minimizer of $L_{\mathrm{y}}(\theta; \mathbf{w}, \lambda)$, but also allows for the use of an almost infinite number of training weights and tuning parameters during the training step, thereby avoiding over-fitting.

2.3 Illustration with a simple example

A novel aspect of our formulation is represented by the minimization of the integrative loss (3), which combines the individual optimization step required by each classical replication with the approximation of the functional form G. Let us consider the bootstrap procedure for a toy linear regression example with data (y_i, X_i), $i = 1, \ldots, n$, and the loss function $\ell(\theta; y_i, X_i) = (y_i - X_i^\top \theta)^2$ and $\lambda = 0$. For this problem, we can obtain the closed-form solution of the optimization problem for each bootstrapped sample: $G_0(\mathbf{w}) = (\mathbf{X}^\top W \mathbf{X})^{-1} \mathbf{X}^\top W \mathbf{y}$, where

$\mathbf{y} = (y_1, \ldots, y_n)^\top$, $\mathbf{X} = (X_1^\top, \ldots, X_n^\top)^\top$, and $W = \mathrm{diag}(\mathbf{w})$. Thus, a bootstrap procedure would follow simple steps: for $b = 1, \ldots, B$, generate $\mathbf{w}^{(b)} = (w_1^{(b)}, \ldots, w_n^{(b)}) \sim \mathrm{Multinom}(n, 1_n / n)$ or $n \times \mathrm{Dirichlet}(n, 1_n)$, and then for each $\mathbf{w}^{(b)}$, plug in the formula to get $\hat{\theta}^{(b)} = G_0(\mathbf{w}^{(b)})$. However, if one does not have the closed-form formula but has to solve numerically the minimization problem of (1) for every generated $\mathbf{w}^{(b)}$, the bootstrap procedure can be prohibitively demanding in computation. Thus, our GMS formulation via (3) can be thought of as an automatic way to find a highly accurate approximation to the closed-form solution (in the form of a neural network) of the minimization problem of (1). Once this solution $\hat{G}$ is found, one can easily generate bootstrap estimators with almost no computational cost.

For a case of *n* = 100 and $p = 10$, we set the true coefficient $\theta = \{1, 0, \ldots, 0\}$ and the regression variance one. The predictors are independently generated from $N(0, I_p)$. We generate a data set and evaluate random weight bootstrap estimators with $\mathbf{w} \sim n \times \mathrm{Dirichlet}(n; 1_n)$, and then numerically evaluate the average loss of (1) on various weights from the trained generator for the classical machine learning approach with *B* = 500 and *B* = 5, 000, as well as the GMS. We initialize the optimization in different five points for each procedure.

In Figure 1, we consider two performance measures for this example: the training loss specified in (5) and the *integrative prediction loss* (IPL) that can be defined as $\mathbb{E}_{\mathbf{w}} \left\| \hat{\theta}_{\mathbf{w}} - g(\mathbf{w}) \right\|^2$. The IPL is approximated by using 1, 000, 000 Monte Carlo evaluations, and the loss values are multiplied by *n* to adjust for the scale of $\mathrm{Var}(\hat{\theta})$. Note that the GMS trains its generator *G* by minimizing the integrative loss (3), whereas the naive generator *g* is trained using the l_2-loss in (5) with *B* = 500 and 5, 000 training samples, respectively. As expected, Figure 1(a) shows that the training l_2-losses for the naive procedures are significantly lower than those for the GMS. However, the IPLs of the considered methods behave quite differently. The naive minimizers (for the cases with *B* = 500 and 5, 000) first decrease their IPLs rapidly, but after 200 iterations their IPLs begin to increase. In contrast, the GMS seamlessly reduces its IPL. The poor predictive performance of the naive procedure stems from

the fact that the l_2-loss encourages the generator function $\hat{g}$ to overfit the training set $\hat{\theta}^{(1)},\ldots,\hat{\theta}^{(B)}$. Unlike the conventional machine learning modeling, the GMS is quite resistant to overfitting, as we can sample $\mathbf{w}$'s at near-zero computational cost during the training of the generator function.

3 Generative Bootstrap Samplers

3.1 Bootstrap and subgroup bootstrap

The simplest use of the GMS is to bootstrap M-estimators, which is a special case of form (3) without η and $u(\cdot)$. The weight distribution is $\text{Multinom}(n, 1_n/n)$ (or $n \times \text{Dirichlet}(n, 1_n)$ for the Bayesian bootstrap). More precisely, we let ϕ be the parameter underlying the generator G and solve the optimization problem: $\hat{\phi} = \text{argmin}_{\phi\in\Phi}\mathbb{E}_{\mathbf{w}}\left[\frac{1}{n}\sum_{i=1}^{n} w_i \ell(G_\phi(\mathbf{w}); y_i)\right]$. We call this simple GMS application the *Generative Bootstrap Sampler* (GBS).

Despite its considerable efficiency, the GBS framework has a fundamental limitation for practical bootstrap applications: the dimension of the generator domain equals the sample size n. Even when computationally efficient neural networks are used to model the generator, the convergence is quite slow when the input dimension is high (say, tens of thousands). We may further encounter technical issues such as memory shortage as well, which is particularly severe for big data. To address this limitation, we consider a subgroup weighting strategy, which divides the data set into subgroups and assigns equal weights to observations within each subgroup. The subgrouping idea is primarily used for bootstrapping time series data sets, referred to as *block bootstrap* (Lahiri, 1999; Härdle et al., 2003), in order to preserve the temporal association within bootstrapped samples. In contrast to the time series applications, we use subgrouping (or blocking) to reduce the number of weights, or more precisely, the domain dimension of the generator function so as to save computational costs.

Let $[n]$ denote the index set $\{1,\ldots,n\}$ of the observations. We consider an exclusive and exhaustive partition: $I_1,\ldots,I_S \subset [n]$ such that $I_i \cap I_j = \emptyset, \forall i \neq j$, and $\cup_{s=1}^{S} I_s = [n]$.

Without loss of generality, we assume that the size of each I_s is the same, i.e., $|I_s| = n/S$ for $s = 1, \ldots, S$. We define a subgroup assignment function $h : [n] \mapsto [S]$ such that $h(i) = s$ if $i \in I_s$. Then, for $\{\alpha_1, \ldots, \alpha_S\}^{\mathrm{T}} \sim \mathbb{P}_\alpha$, with $\mathbb{P}_\alpha$ being an S-dimensional weight distribution, we impose the same value of weight on all elements in a subgroup as

$$w_i = \alpha_{h(i)} \quad \text{for } i = 1, \ldots, n. \qquad (6)$$

and we denote $\mathbf{w}_\alpha = \{\alpha_{h(1)}, \ldots, \alpha_{h(n)}\}^{\mathrm{T}} \in \mathbb{R}^n$. As a result, it follows that $\alpha_{h(i)} = \alpha_{h(k)}$, if $i, k \in I_s$ for some s. Similar to the vanilla GBS, setting $\boldsymbol{\alpha} \sim \mathrm{Multinomial}(S, 1_S / S)$ or $\boldsymbol{\alpha} \sim S \times \mathrm{Dirichlet}(S, 1_S)$ result in the block-based nonparametric bootstrap and Bayesian bootstrap, respectively.

As an illustration, we consider a simple linear regression example by generating a data set from the model with n = 1000, p = 10 and the coefficients θ being a sequence of equi-spaced values between –2 and 2. Each covariate is drawn i.i.d. from $N(0, 1)$, and the regression variance is set to one. The resulting domain dimension of a vanilla G is 1000. Figure 2 shows individual histograms of bootstrap distributions with varying subgroup sizes. Even when the number of subgroups is tiny (S = 5), the obtained bootstrap distributions are acceptable, although the variability tends to be underestimated. As S increases (S = 25), the quality of the approximation of the subgroup bootstrap distribution improves significantly. When S = 100, the subgroup bootstrap distributions are indistinguishable from the target ones. When we use 100 subgroups (10 observations in each subgroup), the input dimension is reduced to 100 from the original 1000 but the resulting bootstrap distributions are nearly identical to those from the standard bootstrap (see Figure 3). We use S = 100 by default.

Remark. Under some regularity conditions, one can show that the subgroup bootstrap is consistent when S is of a higher order than $\sqrt{n}$ (see the Supplementary Materials for a formal proof).

3.2 Iterated bootstrap

The iterated bootstrap method was proposed to improve the inference accuracy of the simple bootstrap method, and was shown both theoretically and empirically to achieve a higher-order accuracy for the coverage of the constructed confidence intervals and bias-corrections Martin, 1992; McCarthy et al., 2018; Hall, 2013; Lee and Young, 1999, 1995. More precisely, an iterated bootstrap procedure involves nested levels of data resampling.

The double bootstrap, which is the simplest iterated bootstrap, first creates B bootstrap samples, $\mathbf{y}_b^*$, for $b = 1,\ldots,B$ by resampling from the original data set, and then, for each bootstrapped sample $\mathbf{y}_b^*$, creates C second-level bootstrap samples, $\mathbf{y}_{bc}^{**}, c = 1,\ldots,C$, by resampling from $\mathbf{y}_b^*$. For each $\mathbf{y}_b^*$ and $\mathbf{y}_{bc}^{**}$, we denote the corresponding estimator of θ by $\hat{\theta}_b^*$ and $\hat{\theta}_{bc}^{**}$, respectively. By iterating this step, we can simply extend this to more iterated bootstrap cases.

Various procedures for constructing confidence intervals using bootstrap have been proposed, such as the *percentile* method (Hall, 1992), the *studentized* method (Hall, 1988; Efron, 1979), the *Bias-Corrected and accelerated* method BC_a (Efron, 1987), and *Approximated Bias Correction* (ABC; Diciccio and Efron (1992)), etc. Even though BC_a and ABC procedures enjoy the second-order accuracy (fast convergence in coverage error), a practical implementation of these procedures are not trivial since it is difficult to calculate their acceleration factor for general models. On the other hand, the percentile procedure is only first-order correct, and the studentized procedure requires an iterated bootstrap unless an explicit form of the standard error of the bootstrap estimator is available. To improve the quality of the constructed CI, we consider using double bootstraps as in the coverage calibration method (Hall and Martin, 1988; Hall, 1986) and studentized CI procedure (Hall, 1988). The calibrated percentile two-sided CI via double bootstrap achieves the second-order accuracy $O(n^{-1})$, while its single bootstrap counterpart only attains a rate of $O(n^{-1/2})$. However, applying the conventional double bootstrap requires undesirably intensive computation: a total of $B \times C$ evaluations of bootstrap estimators $\hat{\theta}_{bc}^{**}$ for $b = 1,\ldots,B$ and $c = 1,\ldots,C$. Lee and Young (1999) showed that B and C should be of a higher order than n^4 and n^2 for two-sided CIs and of order n^2

and n for one-sided CIs, respectively, so that the coverage error rate of the Monte Carlo interval is no greater than that of the theoretical double bootstrap interval. The authors considered B = 1000 and C = 500 in their simulations, resulting in a total of 500, 000 evaluations, which is an unmanageable size under the conventional bootstrap framework.

3.3 GBS for iterated bootstrap

Extending the GBS to iterated bootstraps is immediate as it is a special case of (3) with a weight distribution that has a hierarchical structure. For a d-level iterated bootstrap procedure, we may characterize its weight distribution hierarchically: $\mathbf{w}_{(1)} \sim \text{Multinom}(n, 1_n / n)$, …, $\mathbf{w}_{(d)} \mid \mathbf{w}_{(d-1)} \sim \text{Multinom}(n, \mathbf{w}_{(d-1)} / n)$. The computational advantage of the GBS framework is particularly significant in these iterated situations.

One drawback of the standard nonparametric bootstrap is that each bootstrap sample only touches upon about $1-e^{-1} \approx 63\%$ of the observations due to the nature of multinomial sampling, which appears to be somewhat wasteful. This loss is compounded and become more significant in iterated bootstraps. A smoothed version of these weight distributions is a hierarchy of Dirichlet distributions, which enable each $\hat{\theta}_b^*$ and $\hat{\theta}_{bc}^*$ to utilize all the observations Cheng and Huang, 2010; Xu et al., 2020; Præstgaard and Wellner, 1993. Thus, we mainly consider $\mathbf{w} \mid \mathbf{z} \sim n \times \text{Dirichlet}\ (n, \mathbf{z})$ and $\mathbf{z} \sim n \times \text{Dirichlet}\ (n, 1_n)$. If a subgroup bootstrap as in Section 3.1 is employed the subgrouped weights follow $\mathbf{w} \mid \mathbf{z} \sim S \times \text{Dirichlet}\ (S, \mathbf{z})$ and $\mathbf{z} \sim S \times \text{Dirichlet}\ (S, 1_S)$. We train a generator function that covers both single and double bootstraps by adopting a probabilistic mixture of single and double bootstrap weights distributions; e.g., generate single or double bootstrap weights with 50%-50% chances.

3.4 An illustration: double-bootstrap for logistic regression

We consider the standard logistic regression model: $y_i \sim \text{Bernoulli}\left(\frac{1}{1+\exp\{-X_i^\top \theta\}}\right)$, where $X_i \in \mathrm{R}^p$ and $\theta \in \mathbb{R}^p$ for $i = 1,\ldots,n$. To apply the GBS to this model, we simply set the loss function to be $(1-y_i)X_i^{\mathrm{T}}\theta + \log(1+\exp(-X_i^{\mathrm{T}}\theta))$ in (3). We simulate a data set that contains n = 400 observations, each with p = 20 covariates generated independently from the standard Gaussian. The true coefficient vector is set to be an equi-spaced sequence between –3 and 3.

We examine 95% CIs constructed by various procedures, including a bias-corrected percentile CI (single bootstrap, denoted by "basic"), a naïve percentile CI (single bootstrap, denoted by "Percentile"), a calibrated percentile CI (double bootstrap), and a studentized CI (double bootstrap). The "basic" CI is constructed as $(2\hat{\theta} - q^*_{97.5\%}, 2\hat{\theta} - q^*_{2.5\%})$, where q^*_β is the β-quantile of the bootstrap distribution of $\hat{\theta}^*$. The calibrated percentile CI is obtained as $(2\hat{\theta} - q^*_{\hat{\alpha}_U}, 2\hat{\theta} - q^*_{\hat{\alpha}_L})$, where $\hat{\alpha}_L$ and $\hat{\alpha}_U$ are calibrated coverage levels via the double bootstrap aiming at 2.5% and 97.5%, respectively. The studentized CI is $(\hat{\theta} - \tilde{t}^*_{97.5\%}\hat{s}, \hat{\theta} - \tilde{t}^*_{2.5\%}\hat{s})$, where $\tilde{t}^*_\beta$ is the β-quantile of the studentized bootstrap statistic, and $\hat{s}$ is the estimated standard error (a detailed description of these bootstrap procedures is given in Section B of the supplementary materials). The coverage is calculated as the proportion of how many individual true parameters are covered by the bootstrap marginal CIs. Figure 4 shows these CIs, which are marked green if they cover the true θ, and in light red if not. Figure 4 shows that, despite the fact that the basic single bootstrapped CI (top left) and the double bootstrapped CIs (bottom left and bottom right) both satisfy the target coverage 95%, the width of the single bootstrap is clearly wider than those of the double (1.64 for the single vs. 1.18 and 1.29 for the double). In addition, Figure 4 also demonstrates that the GBS bootstrap CIs are almost indistinguishable from the classical bootstrap CIs. GBS for percentile bootstrap shares the same poor coverage (80%) as the classical percentile bootstrap (80%), along with nearly identical widths of the CIs (1.64 for GBS vs. 1.67 for the classical bootstrap). The classical bias-corrected percentile bootstrap ("Basic") attains 95% coverage, and so does GBS (Basic) counterpart.

For the double bootstrapped CIs, we generate 5000 bootstrap samples for the first-level and 1000 for the second-level, resulting in a total of $5000 \times 1000 = 5,000,000$ bootstrap evaluations. This poses a significant computational challenge under the conventional framework. In comparison, once the generator function is trained (which takes less than 3 minutes for this example), the GBS produces 10, 000 bootstrap estimators in less than 0.1 second, and its computational advantage is even more significant when n and p are larger, as shown next.

3.5 Scaling up towards large *n* and *p*

We consider the same logistic regression model as in Section 3.4, and the true regression coefficients $\{\theta_j\}$ is set to be an equi-spaced sequence in $(-c, c)$, where the value of c is chosen to match the in-sample classification error to 5% or 10%. We compare the performance of the GBS with those of the standard bootstrap, BCa (Efron, 1987), Wald interval and the profile likelihood confidence interval with sample size $n \in \{500, 5000, 10000\}$ and dimension of covariates $p \in \{30, 200, 300\}$. This simulation is replicated independently 20 times. We examine properties of the 95% CIs constructed by these bootstrap methods (i.e., the average coverage and average width, and their actual computing time). For standard bootstrap procedures, we consider both a parallel computing environment using 25 CPU cores (abbreviated as "25C"), and a single-core computation (i.e., "1C"). The detailed setting is described in Section 6.3, and the specification of the computing server is given in the the supplementary materials. We use the `R` package `boot` to implement conventional bootstrap procedures. The classical Wald CI based on Fisher information is obtained for comparison. The profile likelihood CI is based on an asymptotic approximation, and its computation is carried out by using the `confint` function in `R`. Due to the computational burden, the conventional CI procedures for large sized data sets are too expensive, so we only report the estimated computation times using two replicates.

Table 1 and Table 2 compare traditional bootstrap procedures with their GBS equivalents in various settings. The GBS procedures are comparable to their

conventional counterparts (“Basic” and “Percentile” in the table) in terms of the coverage and width of the constructed CIs. The standard bootstrap percentile CIs (“ Percentile”) have been shown to have low coverage in all simulations. GBS1 (Percentile), a fast approximation to the bootstrap, performs nearly equally badly. For high-dimensional logistic regression, confidence intervals based on asymptotic approximation, such as the profile likelihood and the Wald CI, also have low coverage (lower than the nominal 95%). In contrast, the bias-corrected bootstrap (Basic”) attains very good coverage, and so does its fast approximation, GBS1 (Basic). Appendix D provides more detailed descriptions and analyses, including GBS1 coverages against single bootstrap counterparts per replication. The results show that GBS can recover its original bootstrap results almost perfectly. Additionally, the “Time” column in Tables 1 and 2 reveals that GBS greatly reduces the classical bootstrap’s computing time.

When $(n, p) = (500, 30)$, the traditional bootstrap-based CIs are significantly faster to compute. However, as data size increases, the conventional bootstrap becomes prohibitively expensive, taking more than an hour for $(n, p) = (10000, 300)$ using a parallel computation with 25C, and more than 7 hours using 1C. Due to its heavy computational need, the BCa cannot produce meaningful results for moderately large data sets (e.g., for $(n, p) = (5000, 200)$ and (10000, 300)). The profile likelihood procedure (“Profile”), which is based on an asymptotic approximation of the sampling distribution, is also quite expensive when data size becomes large.

For the double bootstraps, the conventional repetitive computations take more than 2.5 hours with parallel computation using 25C for the case with $(n, p) = (500, 30)$, and would have taken more than 48 days for the case with $(n, p) = (10000, 300)$. As a result, the conventional double bootstrap procedures are infeasible for multiple replicates, so their results are omitted in Table 1 and Table 2. In contrast, the GBS training takes less than three minutes for all examined settings, while the generation and post-processing for the double bootstrap take about one minute. Furthermore, the double-bootstrap GBS2s requires very little extra computational time, but achieves a significantly higher accuracy, than the single bootstrap GBS1s.

4 Bootstrap Cross-Validation for Parameter Tuning Via GMS

Tuning parameter selection has been a challenging and computationally intensive task for many statistical and machine learning algorithms since repetitive computations are often required over a wide range of possible choices of the tuning parameter. We note that the GMS framework is not only applicable to bootstrap, but can also be used to expedite the computation of *Cross-Validation* (CV) procedures. It is easy to see that for a weight $w_i = 0$, the corresponding term in the weighted M-estimation loss function (1) is zero, which is equivalent to ignoring observation y_i. More generally, we denote $\mathbf{w}_{(-I)} = \{w_1, \ldots, w_n\}$ with $w_i = 0$ for $i \in I$, and $\{w_i : i \notin I\} \sim (n - |I|) \times \text{Dirichlet}(n - |I|; 1_{n-|I|})$. Thus, index sets I and I^c can be viewed as those for the test and training data, respectively. To train the CV generator without the bootstrapping aspect, one may employ a simpler weight distribution than the multinomial or Dirichlet, such as setting all the weights in a randomly selected fold to be zero, and the remaining to be one. Based on this setup, a simple modification of Algorithm 1 (with strategies in Section 6.3) can be used to train the generator for the K-fold CV (more details in the Supplementary Materials). Once the generator is trained, one can easily compute the estimated out-of-sample error across different tuning parameters by alternating zero weight for each fold.

More precisely, for $b = 1, \ldots, B$ and a tuning parameter λ_l in a candidate set $\{\lambda_1, \ldots, \lambda_L\}$, we set zero weights on a fold I_k^* for $k = 1, \ldots, K$; i.e., $w_i^{(b,k)} = 0$ for $i \in I_k^*$. For $i \notin I_k^*$, we can set $w_i^{(b,k)} = 1$ when only CV is of interest, or let $\{w_i^{(b,k)}\}_{i \notin I_k^*} \sim (n - |I_k^*|) \times \text{Dirichlet}(n - |I_k^*|, 1_{n-|I_k^*|})$ so as to quantify uncertainty in the CV via bootstrap. The bootstrapped CV estimator without considering the test set I_k^* with a tuning parameter λ_l, denoted by $\hat{\theta}^{(b)}_{(-I_k^*),\lambda_l}$, can be computed as $\hat{G}(\mathbf{w}^{(b,k)}, \lambda_l)$. The CV loss for the k-th fold and λ_l follows as $\hat{e}_l^{(b,k)} = \sum_{i \in I_k^*} \ell(\hat{\theta}^{(b)}_{(-I_k^*),\lambda_l}; y_i) / |I_k^*|$. After repeating this step for all the K folds, we obtain the bootstrapped K-fold CV errors as $\bar{e}_l^{(b)} = \sum_{k=1}^{K} \hat{e}_l^{(b,k)} / K$. After obtaining $\bar{e}_l^{(b)}$ for $l = 1, \ldots, L$ and $b = 1, \ldots, B$, one can easily

identify the bootstrap distribution of the out-of-sample loss via the empirical distribution of $\{\overline{e}_l^{(b)}\}_{b=1,\ldots,B}$ under λ_l, as well as confidence bands of the out-of-sample loss over $\{\lambda_1,\ldots,\lambda_L\}$.

Moreover, with $l^{(b)} = \mathrm{argmin}_l\{\overline{e}_l^{(b)}\}$, the empirical distribution of $\{\lambda_{\min}^{(b)} \overset{\Delta}{=} \lambda_{l^{(b)}}, b = 1,\ldots,B\}$ serves as the bootstrap distribution of the minimizer of CV errors and can naturally quantify the uncertainty of the chosen tuning parameter (an example is given in the left of Figure 6). For example, this bootstrap distribution $\{\lambda_{\min}^{(b)}\}$ provides us an alternative to the *ad hoc* one-standard-error rule commonly recommended for Lasso regression, in which one chooses the most parsimonious model whose CV error is no more than one standard deviate above that of the best model. In contrast, with the availability of the bootstrap distribution of $\lambda_{\min}$, we may pursue a more parsimonious model by using the lower $(1-\alpha)$% confidence bound of this distribution as our chosen λ.

Cross-validation for LASSO and ridge regression. Two representative examples of the penalized M-estimation are ridge (Hoerl and Kennard, 1970) and LASSO regression models (Tibshirani, 1996), with the corresponding loss function for GMS:

$$\mathbb{E}_{\mathbf{w},\lambda}\Big[\frac{1}{n}\sum_{i=1}^{n} w_i\{y_i - X_i^{\mathrm{T}} G(\mathbf{w},\lambda)\}^2 + \lambda u(G(\mathbf{w},\lambda))\Big], \qquad (7)$$

with $u(x) = \|x\|_2^2$ for the ridge regression and $u(x) = \|x\|_1$ for the LASSO. This setting is closely related to the weighted Bayesian bootstrap (WBB) setting analyzed recently in Newton et al. (2021) and Ng and Newton (2022). For this problem the GMS learns the mapping between $(\mathbf{w},\lambda)$ and $\hat{\beta}_{\mathbf{w},\lambda}$, the optimal solution under the WBB setting. After obtaining the trained $\hat{G}$ from (7), for a given input $\mathbf{w}^*$ and λ^*, its output $\hat{G}(\mathbf{w}^*,\lambda^*)$ approximates the minimizer of $\sum_{i=1}^{n} w_i^* \ell(\theta; y_i)/n + \lambda^* u(\theta)$ with respect to θ. We simulated from a linear regression model with n = 500, p = 50, the true parameter $\theta_0 = \{1,-2,1,0,\ldots,0\}$, and $\sigma_0^2 = 1$. Each covariate vector X_i follows iid $N(0,\Sigma)$ with $\Sigma_{kl} = 1$ for $k = l$ and $\Sigma_{kl} = 1/2$ for $k \neq l$.

Figure 5 shows solution-path plots that depict the relations between the tuning parameter choices and the corresponding estimated ridge and LASSO estimators. The x-axis indicates the l_2 norm of the ridge regression or l_1 norm of the LASSO estimators based on a series of λ's, and the y-axis, the value of the estimated coefficient. After the generator is trained by minimizing (7), ridge (top left) and LASSO (bottom left) coefficient values are simply $\hat{G}(1,\lambda)$, which generates the curves in Figure 5 by letting λ vary from 0.0006 to 0.6. The resulting solution-paths of the GMS ridge and LASSO procedures show that the proposed method approximates the standard ones obtained by LARS (Efron et al., 2004) very accurately.

We further investigate how the GMS-bootstrap helps to quantify uncertainty in choosing λ. Figure 6 illustrates some benefits of the bootstrapped CV procedure for the LASSO example. The left panel shows a 95% confidence band for the CV errors across λ. As Efron and Tibshirani (1997) noted, the bootstrapped CV improves the performance of prediction error estimation. However, due to heavy computational burden in the standard bootstrap algorithm, applications of the bootstrapped CV have been greatly hindered. The example in Figure 6 shows that the GMS helps overcome this computational difficulty. The center panel depicts the WBB distribution of the minimizer λ of the CV errors (the red line is the estimated density function). If the CV error curve is of main interest, one can easily generate it by the GMS using binary weights (corresponding to the chosen and left-out folds) as the input. In the right panel of Figure 6, the CV error curve obtained by the standard CV computation is nearly identical to that by the GMS.

5 Quantile Regression Inference at Various Quantile Levels

Quantile regression models, which assume that a certain quantile of the response variable linearly depends on the covariates, have been commonly used for robust regression analysis (Yu et al., 2003; Yu and Moyeed, 2001; Koenker, 2004). More precisely, for a given $\eta \in (0,1)$, the conditional η-th quantile of the response given X_i is modeled by $X_i^{\mathrm{T}}\theta$. The standard loss function for fitting such a model is

$$\ell(\theta; y_i, X_i) = \rho_\eta(y_i - X_i^{\mathrm{T}}\theta), \quad (8)$$

where $\rho_\eta(u) = (\eta - I(u < 0))u$. The inference for the regression coefficients in this setting is more challenging than that for parametric regression models, because the sampling distribution of the coefficient estimates often relies on the regression error density function, which needs to be estimated and is a challenging task by itself in high-dimensional settings (Koenker, 1994). In routine applications of quantile regression analyses, bootstrap procedures are popular to use for approximating the sampling distribution of the estimates (Feng et al., 2011; Hahn, 1995; Kocherginsky et al., 2005), which can be computationally demanding. Furthermore, when a practitioner is interested in investigating multiple quantile levels, it is also necessary to repeat the bootstrap procedure multiple times, each at a different quantile level. Such a computational burden is prohibitive when the data size is large.

By using $\ell(G(\mathbf{w}, \eta); y_i, X_i) = \rho_\eta(y_i - X_i^\top G(\mathbf{w}, \eta))$ in (3), we apply the GMS to overcome the computational challenges for the inference of quantile regression models with a GMS loss of

$$\hat{G} = \underset{G}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{w},\eta}\Big[\sum_i^n w_i \rho_\eta(y_i - X_i^\top G(\mathbf{w}, \eta))\Big], \quad (9)$$

where $\mathbb{E}_{\mathbf{w},\eta}$ is the expectation operator on $\mathbf{w}$ and η, assuming that η follows some distribution $\mathbb{P}_\eta$ whose support is (0,1) and independent with $\mathbf{w}$. A default choice is to add random noises to the candidate set of quantile levels, and let $\mathbf{w}$ follow the probability law in (6).

To demonstrate the effectiveness of this procedure, we test the method on a simulation setting examined in Feng et al. (2011). The data set is generated from the model $y_i = X_i^{\mathrm{T}}\theta_0 + 3^{-1/2}[2 + \{1 + (x_{1i} - 8)^2 + x_{2i}\}/10]\epsilon_i, i = 1, \ldots, n$, where $X_i = (x_{i1}, \ldots, x_{ip})^\top$, n = 500, p = 5, $\theta_0 = 1_5^\top$, and $\epsilon_i \sim t_3$. We let $x_{2i} = 1$ for $i \le 400$ and $= 0$ for $i >$ 400, and generate the other covariates independently from the standard log-normal distribution.

As in Feng et al. (2011), we consider the wild bootstrap, as well as the standard bootstrap. Figure 8 (a)–(c) compare the 90% confidence bands of several coefficients generated by the GMS with those obtained by the standard bootstrap and the wild bootstrap over quantiles varying from 0.05 and 0.95, showing that the the approaches result in nearly identical bands.

To investigate computational efficiency of the GMS for quantile regression, we increase the sample size and the number of predictors in the above simulation model to $(n, p) = (1000, 50)$, (2000, 100), (3000, 150), and (5000, 300), respectively, and consider quantile levels varying from 0.05 to 0.95 with a skip of 0.05 (total 19 quantile levels). We set the first five coefficients of θ_0 to be one and the others be zero. Our target is to obtain 5, 000 bootstrap samples under each setting. Due to heavy computational burden of the standard bootstrap procedure, we compute only five bootstrap evaluations and report an estimated time from them (e.g., multiplying 1, 000 to the time taken for the five evaluations). Figure 8 (a) depicts the computation time required for each procedure. While the GMS can be trained in less than 10 minutes for moderately large data size ($n = 5000, p = 300$), the standard bootstrap requires more than 30 minutes for the smallest data set ($n = 1000, p = 50$) and about 3 months for the case of $(n = 5000, p = 300)$.

6 Computational Strategies for Training the Generator

6.1 Multilayer perceptron

Neural networks have been shown effective for approximating functions with complicated structures. Recently, researchers have experimented with various novel ways of using neural networks, such as constructing generators of real-life-like images and creating generative adversarial networks for approximating high-dimensional distributions (Ledig et al., 2017; Wang et al., 2018; Karras et al., 2018; Goodfellow et al., 2014; Arjovsky et al., 2017). The simplest neural network structure is a class of MLPs/FNNs constructed by composing activated linear transformations. For $k = 1, \ldots, K$, let g_k denote the feed-forward mapping represented by $N^{(k)}$ hidden nodes, where $g_k : \mathbb{R}^{N^{(k)}} \mapsto \mathbb{R}^{N^{(k+1)}}$ is defined as

$g_k(\mathbf{X}) = \sigma(\mathbf{U}^{(k)}\mathbf{X} + \mathbf{b}^{(k)}) \in \mathbb{R}^{N^{(k+1)}}$, where $\mathbf{X} \in \mathbb{R}^{N^{(k)}}$ is the input variable of g_k. Also, this function is characterized by a "weight" parameter and a "bias" parameter: the $N^{(k+1)} \times N^{(k)}$ weight matrix $\mathbf{U}^{(k)}$ and the $N^{(k+1)}$-dimensional bias vector $\mathbf{b}^{(k)} = \{b_1^{(k)}, \ldots b_{N^{(k)}}^{(k)}\}$. A *K*-layer MLP function $g : \mathbb{R}^{N^{(1)}} \mapsto \mathbb{R}^D$ can be defined by the composition of these functions as

$$g(\mathbf{X}) = L \circ g_K \circ \ldots \circ g_1(\mathbf{X}), \qquad (10)$$

where $L : \mathbb{R}^{N^{(K)}} \mapsto \mathbb{R}^D$ is a linear function that maps the final hidden layer $g_K \circ \ldots \circ g_1(\mathbf{X})$ to the *D*-dimensional output space of *g*. Commonly used activation functions include the sigmoid function, the hyperbolic tangent function, the Rectified Linear Unit (ReLU) (Nair and Hinton, 2010), the Exponential Linear Unit (Clevert et al., 2015), the Gaussian Error Linear Unit (Hendrycks and Gimpel, 2016), etc. We here employ neural networks with the ReLU activation function $\sigma(t) = \max\{t, 0\}$ to construct generator *G* in (3) in a novel way as characterized by the integrative loss (3) and the weight multiplicative MLP explained below.

6.2 Weight multiplicative MLP

Despite its generalizability and practicability, we observe that the simple MLP converges slowly for our GMS applications (as shown in Figure 9). We propose a modification motivated by the Taylor approximation of the first derivative of the weighted loss function. For illustration, let us consider the weighted M-estimation loss $\sum_{i=1}^{n} w_i \ell(\theta; y_i)$ and its optimizer $\hat{\theta}_{\mathbf{w}}$ in (1) for a case of p = 1 (ignoring η and λ for simplicity). Under mild conditions, we assume that $\sum_{i=1}^{n} w_i \ell'(\hat{\theta}_{\mathbf{w}}; y_i) = 0$, where ℓ' is the first derivative of ℓ with respect to θ. Then, by using a Taylor approximation of ℓ' at a local region of some arbitrary $g(\mathbf{w})$, we obtain that

$$0 = \sum_{i=1}^{n} w_i \ell'(\hat{\theta}_{\mathbf{w}}; y_i) \approx \sum_{i=1}^{n} w_i \ell'(g(\mathbf{w}); y_i) + \sum_{i=1}^{n} w_i \ell''(g(\mathbf{w}); y_i)(\hat{\theta}_{\mathbf{w}} - g(\mathbf{w})), \qquad (11)$$

where $\ell''(\theta, y)$ denotes the second derivative of ℓ with respect to θ.

The approximation term used in equation (11) contains two different kinds of approximations: the approximation of $\hat{\theta}_{\mathbf{w}} \approx g(\mathbf{w})$ and the Taylor's approximation for $\sum_{i=1}^{n} w_i \ell'(\hat{\theta}_{\mathbf{w}}; y_i)$. The first kind of approximation can be justified by the universal approximation theorem for neural networks Hornik et al., 1989; Barron, 1993; Lu et al., 2017; Kratsios and Papon, 2022. The universal approximation theorem states that a feed-forward neural network is capable of approximating any continuous function, if the size of the neural network is large enough.

Thus, we have

$$\hat{\theta}_{\mathbf{w}} \approx g(\mathbf{w}) - \sum_{i=1}^{n} \frac{w_i \ell'(g(\mathbf{w}); y_i)}{\sum_{j=1}^{n} w_j \ell''(g(\mathbf{w}); y_j)} \stackrel{\Delta}{=} g(\mathbf{w}) + \sum_{i=1}^{n} w_i h_i(\mathbf{w}). \qquad (12)$$

Motivated by this approximation, we propose a new neural network structure called the *Weight Multiplicative MLP* (WM-MLP) as the sum of a simple MLP and a weight multiplicative one:

$$G(\mathbf{w}, \lambda, \eta) = \underbrace{L_1 \circ B_K(\mathbf{w}, \lambda, \eta)}_{\substack{\text{Simple MLP:} \\ g(\mathbf{w})}} + \underbrace{L_2 \circ (\{f \circ B_K(\mathbf{w}, \lambda, \eta)\} \odot \mathbf{w})}_{\substack{\text{Weight multiplicative network:} \\ \sum_{i=1}^{n} w_i h_i(\mathbf{w}, \lambda, \eta)}}, \qquad (13)$$

where "$\odot$" indicates an element-wise multiplication operator; $L_1 : \mathbb{R}^H \mapsto \mathbb{R}^p$ and $L_2 : \mathbb{R}^n \mapsto \mathbb{R}^p$ are linear functions; $B_K : \mathbb{R}^{n+1+1} \mapsto \mathbb{R}^H$ and $f : \mathbb{R}^H \mapsto \mathbb{R}^n$ are simple MLPs with K hidden layers and one hidden layer, respectively. For a large n, the subgroup bootstrap in Section 3.1 reduces the dimension of $\mathbf{w}$ and the network size.

To demonstrate the improvement, we compare the performances of WM-MLP and the simple MLP for various sizes of hidden nodes (500, 1000, 2000) and layers (K = 1, 2, 3), for a logistic regression example. The true θs in the simulations are equi-spaced between –0.5 and 0.5 with p = 100 and n = 1000. We train the generator G from ten random initializations and report the average loss values after 30,000 iterative updates for each MLP structure. The results are summarized in Figure 9, demonstrating that for all network sizes the proposed WM-MLP outperforms the

simple MLP uniformly. In comparison to a large-sized MLP with three hidden layers and 2000 neurons, even a small-scale WM-MLP with a single hidden layer and 500 neurons achieves a lower loss, whereas the simple MLP with one hidden layer performs much poorly. For all examples in the paper, we used the WM-MLP with three hidden layers as a default, and observed that the resulting generator function based on the WM-MLP performed satisfactorily.

6.3 Computational strategy in optimization

It is straightforward to optimize the GMS integrative loss (3) because the expectation can be approximated by a few Monte Carlo samples at each iteration. We use a variant of the popular SGD algorithms such as *Adam* (Kingma and Ba, 2014), *AdaGrad* (Duchi et al., 2011), *RMSProp* (Tieleman et al., 2012), etc, to iteratively update the neural net parameters until the algorithm converges. Algorithm 1 summarizes the detailed steps of the GMS. As in (4), this algorithm samples M values of $\mathbf{w}$'s and λ's to approximate the expectation and updates the neural network parameters via SGD. It is not uncommon nowadays for a data set to be extremely large, to the point that the full data size surpasses the memory capacity of the computer in use. Data subsampling would be advantageous in this setting for training the GMS, which partially updates the weights corresponding to the subsampled data in the same spirit as stochastic optimization (Allen-Zhu et al., 2019).

Technical details of the optimization. In all our examples, we use the WM-MLP with three hidden layers and 1,000 hidden neurons in each layer. In `Pytorch`, algorithm Adam is used with a learning rate of 0.0003 and a decay rate of $t^{-0.3}$ by default. We use full samples in the SGD optimization without mini-batches because the data sizes of the examples we considered are manageable. However, when the data size is massive, minibatch subsampling would be necessary.

Algorithm 1 A general algorithm to train the GMS.

- Set $\mathbb{P}_{\alpha,\lambda,\eta}$, S (subgroup size), M (Monte Carlo sample size), and T (total iterations).

• Randomly split the full data into S subgroups, resulting in an index function $h(\cdot)$ in (6).

• Initialize the neural net parameter $\phi^{(0)}$.

• Set t = 0.

while the stop condition is not satisfied or $t < T$ **do**

• Independently sample M values of $\boldsymbol{\alpha}$'s, λ's, and η's from $\mathbb{P}_{\boldsymbol{\alpha},\lambda,\eta}$.

• Consider $L = \frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{n} \alpha_{h(i)}^{(m)} l(G_{\phi^{(t)}}(\boldsymbol{\alpha}^{(m)}, \lambda^{(m)}, \eta^{(m)}); y_i)/n + \lambda^{(m)} u(G_{\phi^{(t)}}(\boldsymbol{\alpha}^{(m)}, \lambda^{(m)}, \eta^{(m)}))$,

where $\boldsymbol{\alpha}^{(m)}$ is the m-th sample of M $\boldsymbol{\alpha}$'s.

• Update $\phi^{(t+1)}$ by using the gradient of L via a SGD step.

• Let $t = t+1$.

end while

Choosing distributions for $\mathbf{w}$, λ, and η. For bootstrap procedures, the distribution of bootstrap weights $\mathbf{w}$ (or $\boldsymbol{\alpha}$) can be easily chosen depending on the practitioner's interest; e.g., $\mathbf{w} \sim \text{Multinomial}(n, 1_n/n)$ or $\mathbf{w} \sim n \times \text{Dirichlet}(n, 1_n)$. When n is excessively large, the dimension of $\mathbf{w}$ can be reduced by the subgroup bootstrapping method in Section 3.1. As a general rule, when n > 500, we recommend considering subgrouping. While our theoretical evidence suggests that $S \succ n^{1/2}$ is optimal (see Section A.1 in Supplementary Materials), empirically setting S to a few hundreds performs well in all situations shown in this paper. By default, S = 100 was used. Choosing the training distributions for λ and η is more arbitrary because usually we have no reference distributions for λ and η unlike the case of $\mathbf{w}$. We may first set candidate sets for λ and η in advance (which can be large in size) and then add some random noises to form mixture distributions. For example, we can generate $\lambda = \exp\{\log \lambda' + \epsilon\}$, where λ' is randomly selected from the candidate set and $\epsilon \sim N(0, \delta)$ with $\delta = 0.2^2$ as default. For the quantile regression example in Section 5, we generate $\eta = \eta' + N(0, 0.03^2)$ with η' randomly selected from a pre-determined candidate set, and then truncated to be in $(0.001, 0.999)$.

Training stopping criteria. In order to judge the convergence in training the generator function, we first set the maximum number of epochs depending on computational resources at hands (our default is 20,000 epochs). In addition to this stopping

criterion, we also consider an *early stopping rule* that has been commonly used in training general neural networks (Heckel and Yilmaz, 2021; Li et al., 2020; Prechelt, 1998) to determine when we stop the optimization algorithm before reaching the maximum number of epochs. Intuitively, we stop the algorithm when the updates do not further reduce the loss value. More specifically, for each epoch t, we evaluate the averaged loss value L^t on epoch t and compare it with those of the previous epochs $\{L^{t-\ell}, \ell = 1, 2, \ldots, k\}$ for some lags. We terminate the SGD algorithm if L^t is within ϵ of a quantile (such as the median) of the previous losses. We recommend to monitor the change of loss values in the previous k=100 epochs, and use the 25th percentile with $\epsilon = 0.01$.

6.4 Limitation of GBS and GMS

Despite the empirical successes of GBS and GMS in various applications examined in this paper, they are not free of limitations. First, unlike the conventional bootstrap procedures, even for a small-sized data set, training the generator function of GBS and GMS requires a certain amount of computation time as minimum in training the generator. Tables 1 and 2 show that the GBS for the smallest data set ($n = 500, p = 30$) takes about 15 times longer computation time compared to the standard bootstrap using 25 cores in parallel. Second, like all other applications of neural network, choosing optimal hyperparameters such as learning rate, widths of networks, the number of neurons, etc, is not systematically justified and somewhat heuristic. However, we find that our default settings for the WM-MLP proposed in Section 6.3 result in accurate approximations for our examples. Third, when the output dimension (the dimension of θ) and the input dimension (the subgroup size) for the generator are high, the resulting computation can be bottlenecked in terms of the computational time and the convergence of optimization. Even though for a case of $(n = 10000, p = 300)$ in Tables 1 and 2, the GBS approximates the target bootstrap estimators well, the convergence of training would be slow under higher dimensional settings. As a result, it would be desirable to consider a larger network to approximate more complicated target function, resulting in even more longer computation time.

7 Conclusion

We propose the GMS as a general computational approximation framework to accelerate repeated calculations for (penalized) weighted M-estimations. The GMS was shown effective for a variety of statistical inference procedures, including bootstrap methods and cross-validations for general M-estimators. We apply the GMS to a variety of models, including LASSO, logistic regression, quantile regression, etc. The GMS performs well in all of the situations we investigated, and the weighted M-estimators generated by the GMS are sufficiently accurate and comparable to the much more computationally expensive traditional solutions for all inference purposes. By lowering the computational barrier associated with repetitious data-splitting or data-sampling processes such as (bootstrapped) CVs and iterated bootstrap, the GMS opens up a new perspective on modern statistics. To date, these approaches have been less noticed and rarely practiced by the statistical community not because they are less valuable, but because their computation cost is prohibitively high. We expect that the GMS will prove to be an effective tool for augmenting the power of statistical models in the era of big data.

Acknowledgement

This work was supported in part by the NSF grant DMS-2015411.

Supplementary Material

R-package: R package for GMS can be found at the following URL:

`https://github.com/shijiew97/GMS`.

Supplementary Material: The Supplementary Material contains proofs of theorems, additional simulation analysis and details of training algorithms. (**.pdf**)

References

Allen-Zhu, Z., Y. Li, and Z. Song (2019). A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pp. 242–252. PMLR.

Arjovsky, M., S. Chintala, and L. Bottou (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pp. 214–223.

Barbe, P. and P. Bertail (2012). *The weighted bootstrap*, Volume 98. Springer Science & Business Media.

Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory 39* (3), 930–945.

Chatterjee, S., A. Bose, et al. (2005). Generalized bootstrap for estimating equations. *The Annals of Statistics 33* (1), 414–436.

Cheng, G. and J. Z. Huang (2010). Bootstrap consistency for general semiparametric m-estimation. *The Annals of Statistics 38* (5), 2884–2915.

Clevert, D.-A., T. Unterthiner, and S. Hochreiter (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* .

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2* (4), 303–314.

Diciccio, T. and B. Efron (1992). More accurate confidence intervals in exponential families. *Biometrika 79* (2), 231–245.

Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research 12* (7).

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics 7* (1), 1–26.

Efron, B. (1987). Better bootstrap confidence intervals. *Journal of the American statistical Association 82* (397), 171–185.

Efron, B., T. Hastie, I. Johnstone, and R. Tibshirani (2004). Least angle regression. *The Annals of statistics 32* (2), 407–499.

Efron, B. and R. Tibshirani (1997). Improvements on cross-validation: the 632+ bootstrap method. *Journal of the American Statistical Association 92* (438), 548–560.

Feng, X., X. He, and J. Hu (2011). Wild bootstrap for quantile regression. *Biometrika 98* (4), 995–999.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680.

Hahn, J. (1995). Bootstrapping quantile regression estimators. *Econometric Theory 11* (1), 105–121.

Hall, P. (1986). On the bootstrap and confidence intervals. *The Annals of Statistics* , 1431–1452.

Hall, P. (1988). Theoretical comparison of bootstrap confidence intervals. *The Annals of Statistics* , 927–953.

Hall, P. (1992). On bootstrap confidence intervals in nonparametric regression. *The Annals of Statistics* , 695–711.

Hall, P. (2013). *The bootstrap and Edgeworth expansion*. Springer Science & Business Media.

Hall, P. and M. A. Martin (1988). On bootstrap resampling and iteration. *Biometrika 75* (4), 661–671.

Härdle, W., J. Horowitz, and J.-P. Kreiss (2003). Bootstrap methods for time series. *International Statistical Review 71* (2), 435–459.

Heckel, R. and F. F. Yilmaz (2021). Early stopping in deep networks: Double descent and how to eliminate it. In *International Conference on Learning Representations*.

Hendrycks, D. and K. Gimpel (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* .

Hoerl, A. E. and R. W. Kennard (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics 12* (1), 55–67.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks 2* (5), 359–366.

Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics*, pp. 492–518. Springer.

Karras, T., T. Aila, S. Laine, and J. Lehtinen (2018). Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Kleiner, A., A. Talwalkar, P. Sarkar, and M. I. Jordan (2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 76* (4), 795–816.

Kocherginsky, M., X. He, and Y. Mu (2005). Practical confidence intervals for regression quantiles. *Journal of Computational and Graphical Statistics 14* (1), 41–55.

Koenker, R. (1994). Confidence intervals for regression quantiles. In *Asymptotic statistics*, pp. 349–359. Springer.

Koenker, R. (2004). Quantile regression for longitudinal data. *Journal of Multivariate Analysis 91* (1), 74–89.

Kratsios, A. and L. Papon (2022). Universal approximation theorems for differentiable geometric deep learning. *The Journal of Machine Learning Research 23* (1), 8896–8968.

Lahiri, S. N. (1999). Theoretical comparisons of block bootstrap methods. *Annals of Statistics* , 386–404.

Ledig, C., L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690.

Lee, S. M. and G. A. Young (1995). Asymptotic iterated bootstrap confidence intervals. *The Annals of Statistics* , 1301–1330.

Lee, S. M. and G. A. Young (1999). The effect of monte carlo approximation on coverage error of double-bootstrap confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 61* (2), 353–366.

Li, M., M. Soltanolkotabi, and S. Oymak (2020). Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 4313–4324. PMLR.

Lu, Z., H. Pu, F. Wang, Z. Hu, and L. Wang (2017). The expressive power of neural networks: A view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6232–6240.

Martin, M. A. (1992). On the double bootstrap. In *Computing science and statistics*, pp. 73–78. Springer.

McCarthy, D., K. Zhang, L. D. Brown, R. Berk, A. Buja, E. I. George, and L. Zhao (2018). Calibrated percentile double bootstrap for robust linear regression inference. *Statistica Sinica 28* (4), 2565–2589.

Nair, V. and G. E. Hinton (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 807–814.

Newton, M. A., N. G. Polson, and J. Xu (2021). Weighted bayesian bootstrap for scalable posterior distributions. *Canadian Journal of Statistics 49* (2), 421–437.

Newton, M. A. and A. E. Raftery (1994). Approximate Bayesian inference with the weighted likelihood bootstrap. *Journal of the Royal Statistical Society: Series B (Methodological) 56* (1), 3–26.

Ng, T. L. and M. A. Newton (2022). Random weighting in lasso regression. *Electronic Journal of Statistics 16* (1), 3430–3481.

Præstgaard, J. and J. A. Wellner (1993). Exchangeably weighted bootstraps of the general empirical process. *The Annals of Probability* , 2053–2086.

Prechelt, L. (1998). Early stopping-but when? In *Neural Networks: Tricks of the trade*, pp. 55–69. Springer.

Rubin, D. B. (1981). The Bayesian bootstrap. *The Annals of Statistics 9* (1), 130434.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. *nature 323* (6088), 533–536.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. R. Statist. Soc. B* , 267–288.

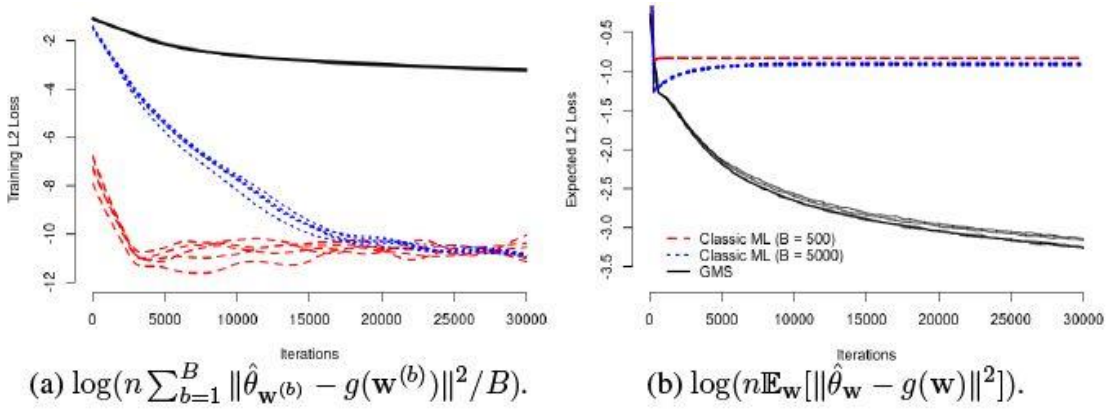
Tieleman, T., G. Hinton, et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning 4* (2), 26–31.

Wang, T.-C., M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro (2018). High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8798–8807.

Xu, L., C. Gotwalt, Y. Hong, C. B. King, and W. Q. Meeker (2020). Applications of the fractional-random-weight bootstrap. *The American Statistician* , 1–21.

Yu, K., Z. Lu, and J. Stander (2003). Quantile regression: applications and current research areas. *Journal of the Royal Statistical Society: Series D (The Statistician) 52* (3), 331–350.

Yu, K. and R. A. Moyeed (2001). Bayesian quantile regression. *Statistics & Probability Letters 54* (4), 437–447.



(a) $\log(n\sum_{b=1}^{B}\|\hat{\theta}_{\mathbf{w}^{(b)}} - g(\mathbf{w}^{(b)})\|^2/B)$. (b) $\log(n\mathbb{E}_{\mathbf{w}}[\|\hat{\theta}_{\mathbf{w}} - g(\mathbf{w})\|^2])$.

Fig. 1 Trace plots of (a) the training loss, and (b) the integrative prediction loss, in the logarithmic scale. Five lines for each optimization represent five distinct initializations; and the red dashed and blue dotted lines indicate the conventional ML with B = 500 and B = 5000, respectively.

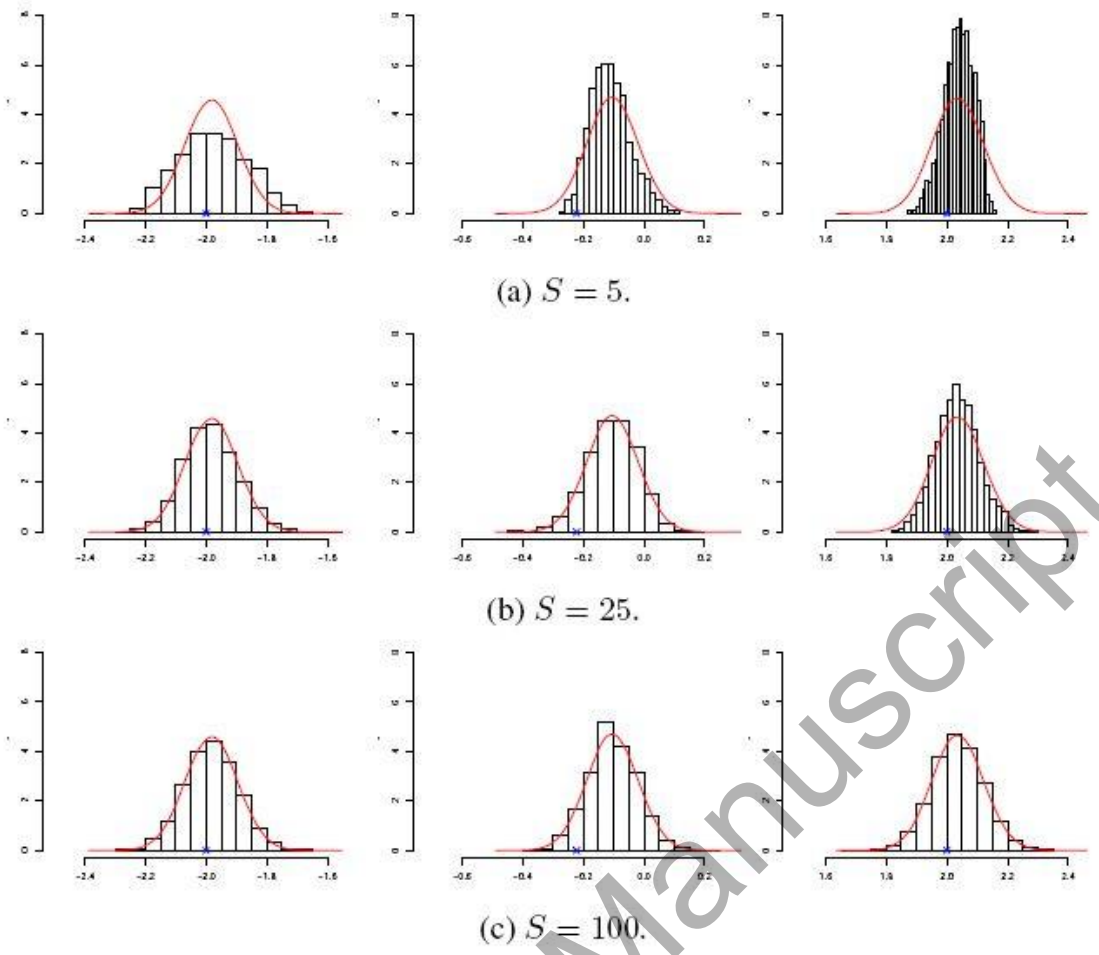


Fig. 2 Histograms of block bootstrap distributions with various S for the coefficient of X_1 (top), X_5 (middle), and X_{10} (bottom) for each subfigure. The red line indicates the density function of the target distribution (of the standard bootstrap).

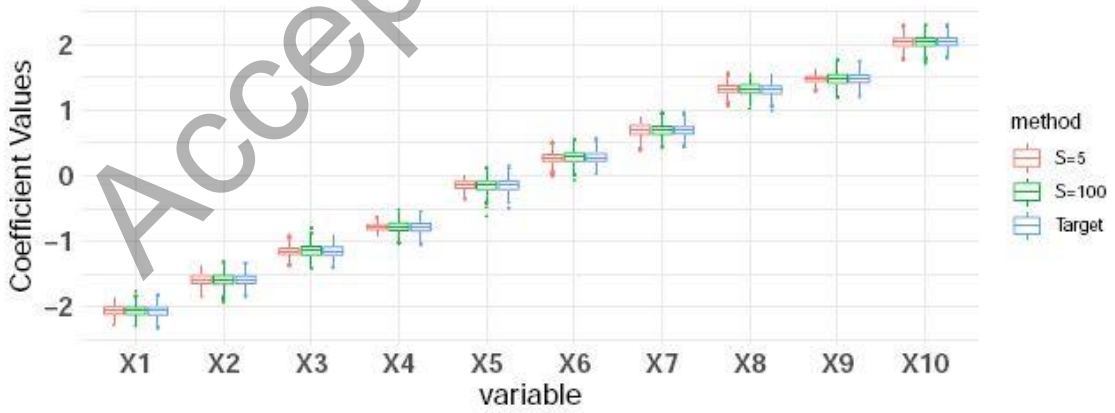


Fig. 3 Comparisons of subgroup bootstraps across different numbers of blocks.

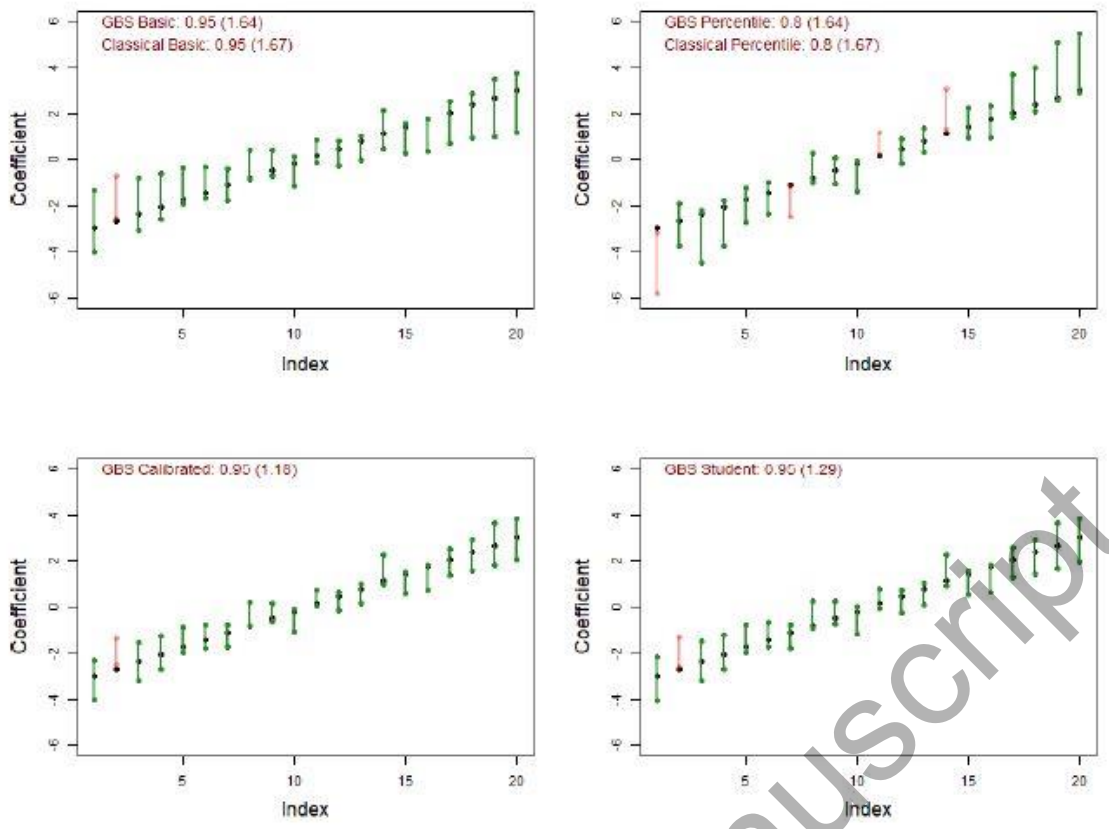


Fig. 4 GBS 95% CIs for the logistic regression example: The basic single bootstrap CI (top left); the naïve percentile single bootstrap CI (top right); the calibrated percentile bootstrap CI via double bootstrap (bottom left); a studentized bootstrap CI via double bootstrap (bottom right). CIs covering true parameters (black dots) are colored in green and otherwise in light red. Averaged width of CIs across parameters is reported in parentheses.

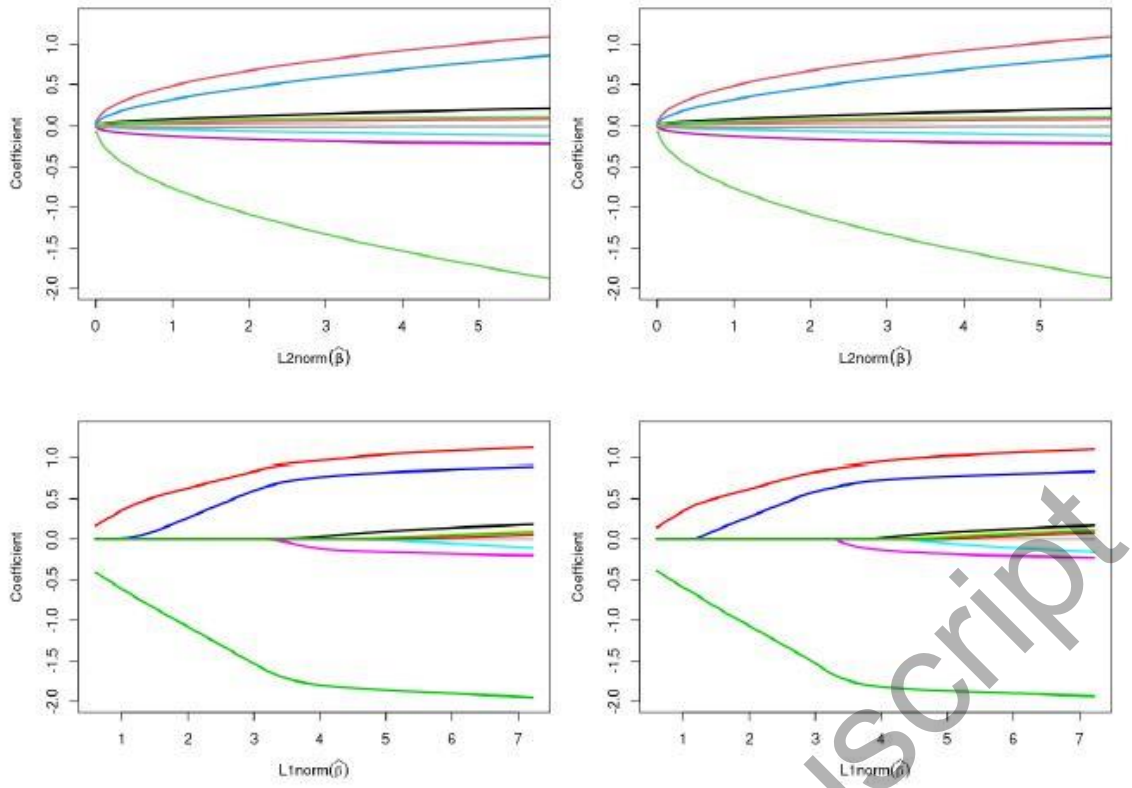


Fig. 5 Solution paths of the GMS ridge (top left), the standard ridge regression (top right) and the GMS LASSO (bottom left), and the LARS (bottom right).

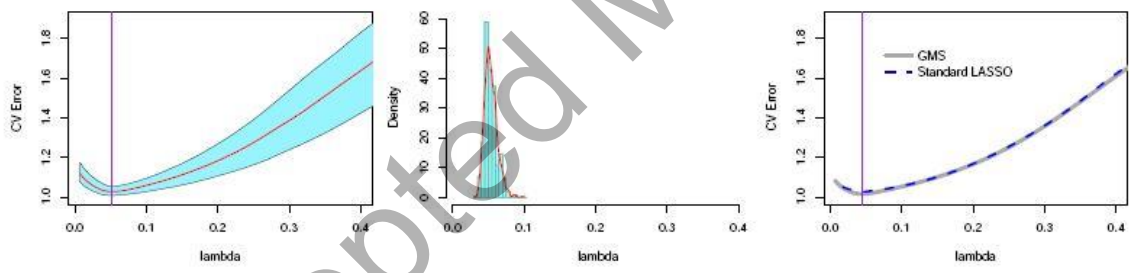


Fig. 6 Left: The 95% confidence band of CV error evaluated from the GMS bootstrap with random weights, and the red solid line indicates the mean curve. Middle: The GMS bootstrapped distribution of the CV-error minimizer $\lambda_{\min}$. Right: CV errors based on the standard LASSO and the GMS with the constant weight vector 1. The purple vertical line indicates the value of λ that minimizes the CV error.

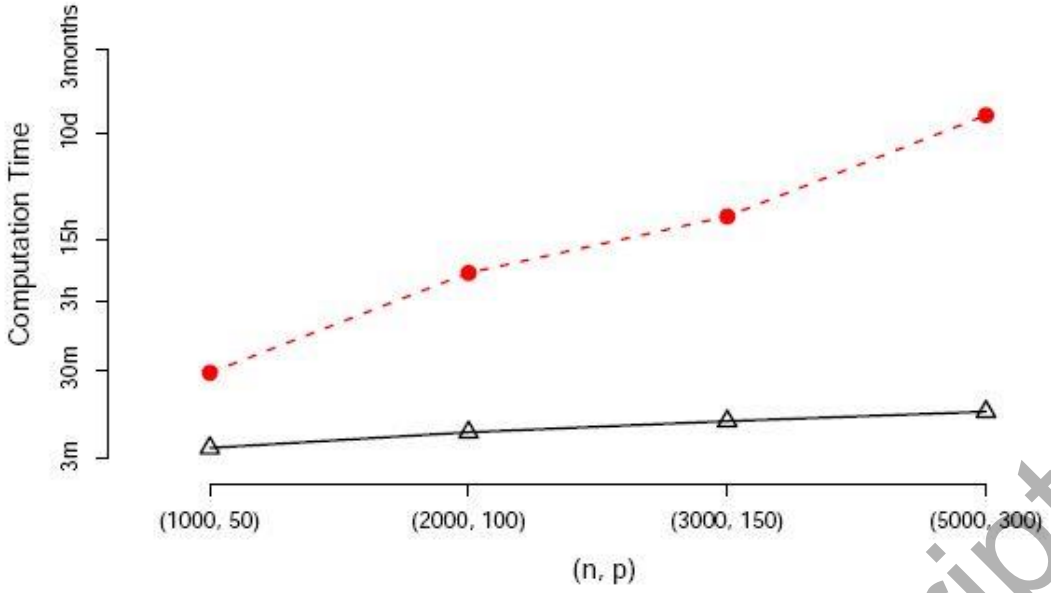
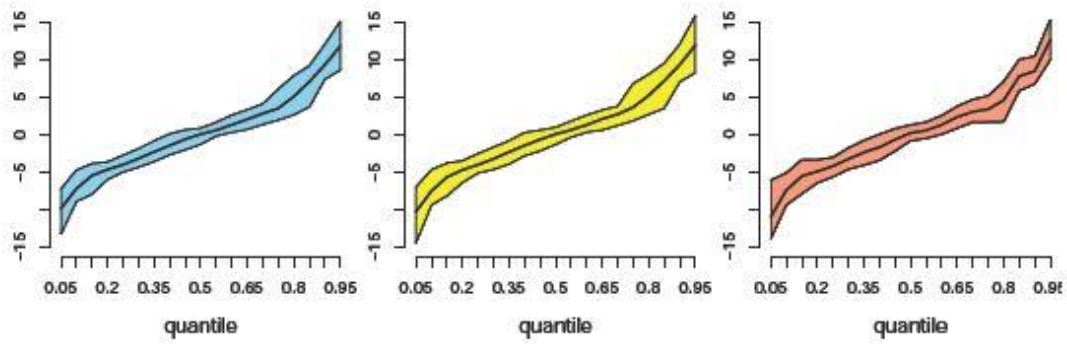
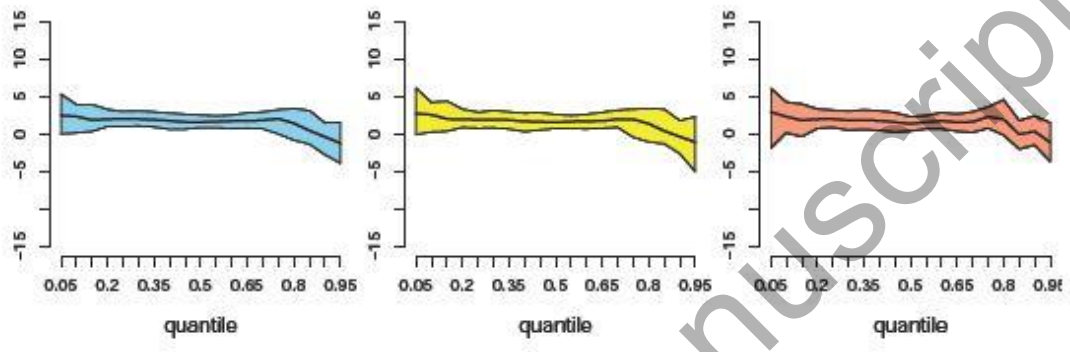


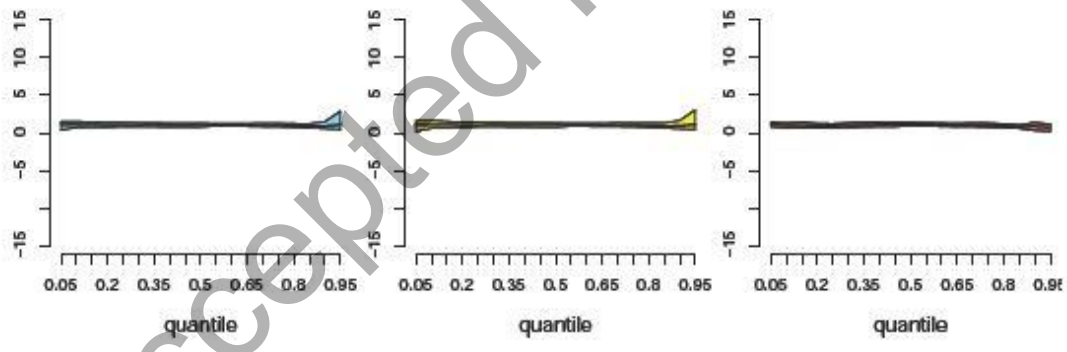
Fig. 7 Computation time for the GMS quantile regression (black solid line with triangle marks) and its conventional counterpart (red dashed line with filled-dot marks).



(a) Intercept



(b) θ_3



(c) θ_5

Fig. 8 (a)–(c): Comparisons between the 90% confidence bands obtained from the GMS (blue), the classical bootstrap (yellow) and the classical wild bootstrap (red) across quantile levels ranging from 5% to 95%. The `quantreg` R package is used for the conventional bootstrap and wild bootstrap.

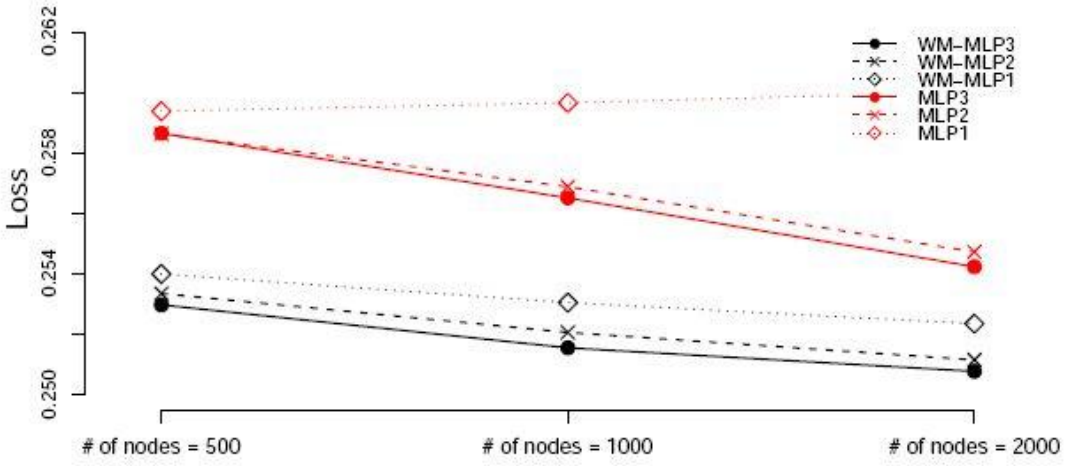


Fig. 9 Comparison of the losses obtained by the simple MLP and the WM-MLP with various numbers of hidden layers and nodes. The number noted after "MLP" indicates the number of layers *K*.

Table 1 Cases of 5% in-sample error in classification. Results of the simulation study for logistic regression models; “GBS1” and “GBS2” indicate single and double bootstraps implemented by the GBS, respectively; “Cov”, “Width”, and “Time” mean the averages (over 20 replicates) of the coverage, the width, and the actual computing time (seconds) of each evaluated 95% CI, respectively; for the computation time of the GBS, the black and red numbers indicate training and generation time (including post processing time for the GBS), respectively.

	$(n, p) = (500, 30)$			$(n, p) = (5000, 200)$			$(n, p) = (10000, 300)$		
Method	Cov	Width	Time	Cov	Width	Time	Cov	Width	Time
GBS1 (Basic)	0.967	2.595	140.8 + 0.1	0.958	0.318	152.9 + 0.2	0.947	0.235	163.6 + 0.4
GBS1 (Percentile)	0.398	2.595	140.8 + 0.1	0.424	0.318	152.9 + 0.2	0.403	0.235	163.6 + 0.4
GBS2 (Student)	0.962	1.762	140.8 + 15.6	0.929	0.298	152.9 + 45.0	0.930	0.225	163.6 + 63.9
GBS2 (Calibrated)	0.927	1.495	140.8 + 15.6	0.924	0.295	152.9 + 45.0	0.929	0.227	163.6 + 63.9
Basic (25C)	0.975	3.677	8.4	0.984	0.36	539.6	NA	NA	4227.05
Basic (1C)			93.8			3833.3			25540.5
Percentile (25C)	0.405	3.677	8.4	0.444	0.36	539.6	NA	NA	4227.05
BCa (25C)	0.818	NA	84.3	NA	NA	NA	NA	NA	NA
Profile	0.678	2.290	0.7	NA	NA	1310.8	NA	NA	8670.7
Wald	0.752	2.253	¡0.1	0.770	0.318	2.5	0.748	0.241	10.8

Table 2 Cases of 10% in-sample error in classification. “Basic” means bias-corrected percentile bootstrap CI and “Percentile” stands for classic percentile bootstrap CI. Results of BCa, profile likelihood CI (“Profile”) and Wald interval (“Wald ”) are also provided. As for classical bootstrap, “25C” represents 25 CPU cores for parallel computing and “’1C’ is a single-core computation.

	$(n, p) = (500, 30)$			$(n, p) = (5000, 200)$			$(n, p) = (10000, 300)$		
Method	Cov	Width	Time	Cov	Width	Time	Cov	Width	Time
GBS1 (Basic)	0.978	0.799	140.8 + 0.1	0.940	0.228	152.9 + 0.2	0.904	0.149	163.6 + 0.4
GBS1 (Percentile)	0.698	0.799	140.8 + 0.1	0.750	0.228	152.9 + 0.2	0.777	0.149	163.6 + 0.4
GBS2 (Student)	0.932	0.711	140.8 + 15.6	0.936	0.210	152.9 + 45.0	0.912	0.153	163.6 + 63.9
GBS2 (Calibrated)	0.885	0.668	140.8 + 15.6	0.932	0.210	152.9 + 45.0	0.882	0.155	163.6 + 63.9
Basic (25C)	0.985	0.844	8.4	0.972	0.228	539.6	NA	NA	4227.05
Basic (1C)			93.8			3833.3			25540.5
Percentile (25C)	0.727	0.844	8.4	0.787	0.228	539.6	NA	NA	4227.05
BCa (25C)	0.972	0.933	84.3	NA	NA	NA	NA	NA	NA
Profile	0.875	0.745	0.7	NA	NA	1310.8	NA	NA	8670.7
Wald	0.890	0.742	¡0.1	0.904	0.216	2.5	0.914	0.148	10.8