Scalable Uncertainty Quantification via Generative Bootstrap Sampler

Minsuk Shin, Lu Wang and Jun S Liu

Abstract

It has been believed that the virtue of using statistical procedures is on uncertainty quantification in statistical decisions, and the bootstrap method has been commonly used for this purpose. However, nowadays as the size of data massively increases and statistical models become more complicated, the implementation of bootstrapping turns out to be practically challenging due to its repetitive nature in computation. To overcome this issue, we propose a novel computational procedure called *Generative Bootstrap Sampler* (GBS), which constructs a generator function of bootstrap evaluations, and this function transforms the weights on the observed data points to the bootstrap distribution. The GBS is implemented by one single optimization, without repeatedly evaluating the optimizer of bootstrapped loss function as in standard bootstrapping procedures. As a result, the GBS is capable of reducing computational time of bootstrapping by hundreds of folds when the data size is massive. We show that the bootstrapped distribution evaluated by the GBS is asymptotically equivalent to the conventional counterpart and empirically they are indistinguishable. We examine the proposed idea to bootstrap various models such as linear regression, logistic regression, Cox proportional hazard model, and Gaussian process regression model, quantile regression, etc. The results show that the GBS procedure is not only accelerating the computational speed, but it also attains a high level of accuracy to the target bootstrap distribution. Additionally, we apply this idea to accelerate the computation of other repetitive procedures such as bootstrapped cross-validation, tuning parameter selection, and permutation test.

1 Introduction

Since its introduction by Efron (1979), the *bootstrap* methods have been considered as a seminal tool in *uncertainty quantification* (UQ) of statistical inference procedures. The bootstrap can be applied to a wide range of situations, of which many lack concrete theoretical guidance and/or other practical means on how to quantify the sampling variability of a statistical procedure. Generally, only some minor regularity conditions, such as a finite variance and a certain degree of smoothness in the functional evaluated, are required for theoretically valid results in bootstrap procedures (Davison and Hinkley, 1997; Efron and Tibshirani, 1994; Hall, 1986, 1992; Shao and Tu, 2012). Despite its popularity and theoretical justifications, the bootstrap's practical use is hindered by its computational burden (and need) of repetitive evaluations. It is required to re-sample the observations and evaluate the statistic at least a few hundreds times, which significantly increase the total computation time. In particular, in the era of *big data*, the size of data sets is

likely to be massive, and statistical models are also highly complicated so that practitioners are not computationally capable of training the statistical model multiple times.

Even when the theoretical form of the sampling distribution of an estimator is explicitly known, a practical UQ would be computationally challenging under high-dimensional settings. For example, in linear regression models with p number of covariates, we know the closed form of the sampling distribution of the coefficient estimator, and its variance is the inverse of the Gram matrix. However, the evaluation of the $p \times p$ inverse matrix takes a computational complexity at an order of $O(p^3)$. When p is large, say millions, this computation is infeasible due to issues of computational time and memory capacity in the computing server (Aghazadeh et al., 2018). In astronomy and computational biology, it is not uncommon to see modern scientific applications where the feature space p is in a dimension of millions (Bray et al., 2016; Vervier et al., 2016; Weinberger et al., 2009; Wood and Salzberg, 2014). For these massive-sized data sets, theoretical sampling distribution itself is not helpful in practical inference such as constructing a confidence interval.

To resolve these issues of UQ for big data and complex models, we propose a scalable computational procedure called *Generative Bootstrap Sampler* (GBS) that accelerates the computation of nonparametric bootstrap (Efron, 1979) and random weight bootstrap (Barbe and Bertail, 2012; Newton and Raftery, 1994; Præstgaard and Wellner, 1993). This novel computational strategy is to construct a generator function that maps the random weights in bootstrap to bootstrap samples of a statistic. This means that once the generator function is trained, one can generate massive-sized bootstrap samples of the statistic with an almost free computational cost. This is implemented by plugging in randomly generated weights to the trained generator.

The proposed procedure enjoys multiple advantages over the existing UQ procedures, and we list them below:

• **Accurate evaluation of bootstrapped distribution**. The GBS is exact or at least approximately accurate. The vanilla version of the GBS is exact in a sense that each bootstrapped statistic generated by the GBS is exactly equivalent to the corresponding counterpart in classical bootstrap sharing the same weight value **w** (the detail of **w** will be introduced in Section 2.1). This result will be rigorously proven in Theorem 2.1. The faster version of the GBS, which is a subgrouped bootstrap (Carlstein et al., 1998) that will be introduced in Section 2.3, is also asymptotically approximating the classical bootstrap distribution as shown in Theorem 2.2. We empirically also show that this approximation is remarkably accurate in various examples such as linear models, logistic regression, Cox proprtional hazard regression, Gaussian mixture models, quantile regression models, etc.

• **Scalability**. The GBS is fast and scalable. We consider *neural networks* to formalize the generator function in the GBS, and its computation can be efficiently implemented via a single optimization procedure, and this optimization is utilizing a subsampling technique of *stochastic gradient descent* (Bottou, 2010). When the sample size is extremely massive, it iteratively updates the generator by using a small subset of samples so that each iteration of optimization is feasible in a GPU computation. We show that the GBS is capable of accelerating the bootstrap by a few hundreds times for various models with massive-sized data sets.

Unlike subsampling-based bootstrap procedures in parallel computing environments (Kleiner

et al., 2014; Politis et al., 2001), the GBS can be used to accelerate the computational speed of high-dimensional problems. The subsampling bootstrap procedures improve computational efficiency in a way that they reduce the sample size via a subsampling and parallelly evaluate bootstrapped estimators. As a result, they suffer from a bias problem by nature and they are not computationally optimal for so-called "large p problems". In contrast, even though the GBS algorithm utilizes subsamples in each iteration, the whole optimization procedure is aiming at minimizing the loss function of the full samples. Not only that, recent machine learning applications to image data sets (Choi et al., 2018; Karras et al., 2018; Ledig et al., 2017; Wang et al., 2018) empirically proved that neural network generators can be efficiently computed even when the dimension of a model is in a scale of millions, and our empirical results also show that the GBS is stably working well when the number of the parameters in the model is as large as thousands.

- **Widely applicable to avoid repetitive computations**. The proposed idea can be generally applied to a wide range of statistical procedures whose loss function can be expressed as a sum of individual losses. The GBS is applicable to general statistical procedures such that i) the loss function of the procedure is differentiable with respect to the parameters of the model, and ii) the generator in the GBS is smooth with respect to the weight values in the loss function of the GBS. Although these two conditions are not necessary to ensure theoretical ideals, they are important in practical and computational performance of the algorithm of the GBS. Since a gradient update is an essential step for the optimization of the neural network in the GBS, the use of SGD for a highly non-differentiable loss function would degrade the optimization stability.

Moreover, we extend the idea of the GBS to other statistical procedures that require multiple computations of similar evaluations. These examples include bootstrapped *cross-validation* (CV; Efron and Tibshirani (1997)) to quantify the uncertainty in the estimation of out-of-sample prediction error, evaluations on difference tuning parameters, and the evaluation for permuted null distribution in hypothesis testing. Like the bootstrap application of the GBS, the GBS offers a way to circumvent multiple repetitive computations in these examples by employing a single optimization on the generator, instead of thousands of computational repetitions. In Section 4, we show that their computational efficiency can be significantly improved by the GBS idea.

We note that while the GBS achieves such desirable properties, one caveat is that its application is restricted to nonparametric bootstrap (Efron, 1979) and weighted bootstrap procedures (Shao and Tu, 2012) in general. Only under limited situations, where an explicit functional form of the data-generating process is available, the GBS can be applied to parametric bootstraps. A brief discussion on this limitation will be given in Section 5.

- **Convenient diagnosis of optimization**. The optimization convergence of the GBS can be diagnosed via an intuitively straightforward way. Since the GBS generates the exact bootstrapped samples of a statistic, the convergence quality of the optimization can be diagnosed by comparing the evaluations from the classical bootstrap procedure and the corresponding GBS counterparts. If the optimization procedure is converged well, these bootstrap samples of the statistic should be the same or close enough to their counterparts. For this purpose, we do not need to evaluate many bootstrap estimator from the classical bootstrap procedure In Section 2.5, we only consider five bootstrap samples to diagnose the convergence, and the computational cost of this evaluation is negligible compared with that of classical bootstrap procedures that require thousands of evalua-

tions.

2   Generative Bootstrap Sampler

2.1   A General Form of Bootstrap

We consider observations $\mathbf{y} = \{y_1, \ldots, y_n\}$ generated from a true data-generating distribution F_θ characterized by a parameter of interest $\theta \in \Theta \subset \mathbb{R}^p$. For many problems, a valid estimator of θ can be found as the solution of the following optimization problem:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^n l(\theta, y_i),$$

with $l(\cdot)$ being a suitable loss function chosen by the researcher. When the class of distributions F_θ belongs to a nice parametric family, an attractive choice of the loss function is the negative log-likelihood and the resulting $\hat{\theta}$ is the *Maximum Likelihood Estimate* (MLE).

The conventional bootstrap procedure for assessing statistical properties of the resulting estimator $\hat{\theta}$ relies on the generation of the *bootstrap samples*, $\hat{\theta}^{(b)}$, $b = 1, \ldots, B$, via the following repetitions

$$\hat{\theta}^{(b)} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^n w_i^{(b)} l(\theta; y_i), \ \ \text{for } b = 1, \ldots, B, \tag{1}$$

where $\mathbf{w}^{(b)} = \{w_1^{(b)}, \ldots, w_n^{(b)}\}$ is independently generated from a distribution $H_{\mathbf{w}}$. More explicitly, for each $\mathbf{w}$ drawn from $H_{\mathbf{w}}$, we can write the resulting solution of (1) as $\hat{\theta}_{\mathbf{w}}$.

The classic nonparametric bootstrap (Efron, 1979) induces the distribution $H_{\mathbf{w}} = \operatorname{Multinomial}(1/n, \ldots, 1/n)$, which is termed "nonparametric bootstrap" and is most boradly used in practice. In contrast, the *parametric bootstrap* procedure generates the bootstrap samples by using repeatedly generated new samples from the estimated distribution in a parametric family, i.e., $F_{\hat{\theta}}$, corresponding to using the negative log-likelihood as the loss function in (1. The Bayesian bootstrap (BB) was introduced by (Rubin, 1981) to provide a nonparametric Bayesian view of the bootstrap, which turns out to be also a smoother version. More precisely, BB can be viewed as the posterior distribution of the unknown sampling distribution under the Dirichlet process prior with its base measure converging to zero. As an extension of the BB, Newton and Raftery (1994) proposes the *Weighted Likelihood Bootstrap* (WLB) by choosing the loss function $l(\cdot)$ as the log-likelihood and employing $H_{\mathbf{w}} = n \times \operatorname{Dirichlet}(1, \ldots, 1)$. Theoretically, they showed that the bootstrap distribution of the WLB is asymptotically valid at a first-order correctness under some regularity conditions. Later the idea was generated to estimating equation frameworks Chatterjee et al. (2005). In this paper, we mainly focus on the WLB, because of technical convenience. The Dirichlet distribution with a uniform parameter of one can be easily approximated by independent exponential distribution. That is, $z_i / \sum_{k=1}^n z_k \sim \operatorname{Dirichlet}(1, \ldots, 1)$ for i.i.d. $z_i \sim \exp(1)$. Since $\sum_{i=k}^n z_k / n \approx 1$ by the law of large number, $n^{-1} \times \{z_1, \ldots, z_n\}$ approximately follows the Dirichlet distribution. This property is convenient in a sense that we do not need to consider the dependence structure in $\mathbf{w}$, and simply generate independent samples from $\exp(1)$.

We note that, although formulation (1) cannot cover all estimators of interest, it is general enough and includes many later variants of bootstrap. All variants of nonparametric bootstrap (Efron, 1979; Kirk and Stumpf, 2009; Rubin, 1981; Silverman and Young, 1987) require repetitions of evaluating the bootstrapped estimators. Even though these evaluations can be completely parallelizable, its computational burden is still hazardously heavy to generate an large enough number of bootstrap samples in practice.

2.2 Generative Bootstrap Sampler

Instead of this inefficient repetition, we take a new point of view at the bootstrap computation. That is, we explicitly treat $\hat{\theta}_{\mathbf{w}}$ as a function of $\mathbf{w}$. By doing so, the bootstrap computation can be summarized as the evaluation of a generator function $G : \mathbb{R}^n \to \mathbb{R}^p$ such that

$$\widehat{G} = \underset{G}{\arg\min} \, \mathbb{E}_{\mathbf{w}} \left[\sum_i^n w_i l(G(\mathbf{w}); y_i) \right], \tag{2}$$

where $\mathbb{E}_{\mathbf{w}}$ is the expectation operator with respect to $\mathbf{w} \sim H_{\mathbf{w}}$. Then, it is straightforward that the distribution of $\widehat{G}(\mathbf{w})$ is equivalent to the conventional bootstrap distribution, and this point is rigorously addressed in the following theorem:

Theorem 2.1. *Suppose that $\widehat{G}$ is the solution of* (2) *and $\widehat{\theta}_{\mathbf{w}} = \arg\min_{\theta} \sum_i^n w_i l(\theta; y_i)$ for a given* $\mathbf{w} = \{w_1, \ldots, w_n\}$. *Also, assume that the solution $\widehat{\theta}_{\mathbf{w}}$ is unique for any $\mathbf{w}$. Then, $\widehat{G}(\mathbf{w}) = \widehat{\theta}_{\mathbf{w}}$.*

Proof. For a given $\mathbf{w}$, we have $\sum_{i=1}^n w_i l(\theta; y_i) \geq \sum_{i=1}^n w_i l(\widehat{\theta}_{\mathbf{w}}; y_i)$ for all $\theta \in \Theta$. This means that $\mathbb{E}_{\mathbf{w}}[\sum_{i=1}^n w_i l(\widehat{G}(\mathbf{w}); y_i)] \geq \mathbb{E}_{\mathbf{w}}[\sum_{i=1}^n w_i l(\widehat{\theta}_{\mathbf{w}}; y_i)]$, which completes the proof. $\square$

In practice, we restrict the generator function G to a class of feed-forwarding neural networks $\mathcal{G}$. As shown in classical references (Barron, 1993; Cybenko, 1989; Hornik et al., 1989), feed-forwarding neural networks flexibly approximate any functional forms when the number of nodes is sufficiently large. Also, applications of image generation in machine learning support the success of neural networks in approximating extremely complicated functions (Arjovsky et al., 2017; Goodfellow et al., 2014; Isola et al., 2017). Under the adoption of the neural networks, the optimization procedure in (2) turns out to minimize the loss function with respect to the weight and bias parameters of the neural network, and this can be implemented via a *backpropagation* algorithm (Rumelhart et al., 1986). Its optimization can be numerically conducted via an efficient GPU-optimized platform such as `Pytorch` and `Tensorflow`. The details of the neural networks are given in Section 2.4.

Once $\widehat{G}$ is trained, one can easily generate bootstrap estimators with almost free computational costs (0.1 seconds to generate 10,000 bootstrap estimators for some examples in Section 3). More detailed steps are described below:

> For $b = 1, \ldots, B$,
> 1. Sample $\{w_1^{(b)}, \ldots, w_n^{(b)}\}^{\mathrm{T}} \sim H_{\mathbf{w}}$.
> 2. Evaluate $\widehat{\theta}^{(b)} = \widehat{G}(\mathbf{w}^{(b)})$.

Although ideally this vanilla version of the GBS generates exact bootstrap samples of a statistic, one critical issue in this procedure is that due to the large-sized input dimension of G, which is
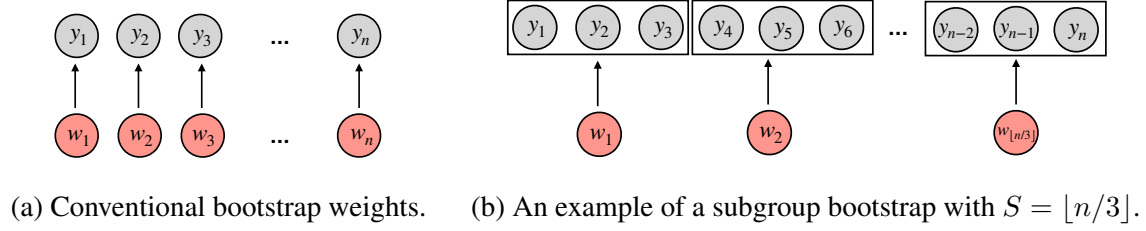
(a) Conventional bootstrap weights.   (b) An example of a subgroup bootstrap with $S = \lfloor n/3 \rfloor$.

Figure 1: The structure of bootstrap weights for the subgroup bootstrap.

n, the optimization of (2) is practically challenging for "large n" data sets. This is because as the dimension of the input increases, the resulting optimization procedures on a high-dimensional neural net tend to converge slowly, and its network size also tremendously increases, which devours vast amounts of computing resources. To relieve this issue, we consider a modified bootstrap procedure that subgroups the observations and imposes the same weight on observations in the same subgroup.

2.3   Subgroup Bootstrap

We consider an arbitrary partition of the index set of observations, $I_1, \ldots, I_S$. That is, $\cup_{s=1}^{S} I_s = \{1, \ldots, n\}$ and $I_i \cap I_j = \emptyset, \forall i \neq j$. Typically $S \to \infty$ as $n \to \infty$ but at a slower rate, i.e., $S/n \to 0$. 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 an index function $u : \{1, \ldots, n\} \mapsto \{1, \ldots, S\}$, which assigns each observation to the corresponding subgroup: if $i \in I_s$ then $u(i) = s$. Then, for some weight distribution $H_{\boldsymbol{\alpha}}$ for $\boldsymbol{\alpha}$, we impose the same value of weight on all elements in a subgroup as

$$w_i = \alpha_{u(i)}, \tag{3}$$

where $\{\alpha_1, \ldots, \alpha_S\}^{\mathrm{T}} \sim H_{\boldsymbol{\alpha}}$ for $i = 1, \ldots, n$, and we denote $\mathbf{w}_{\boldsymbol{\alpha}} = \{\alpha_{u(1)}, \ldots, \alpha_{u(n)}\}^{\mathrm{T}}$. Similar to the vanilla GBS previously introduced, setting $H_{\boldsymbol{\alpha}} = S \times Dir(1, \ldots, 1)$ and $H_{\boldsymbol{\alpha}} = Multinomial(1/S, \ldots, 1/S)$ result in an approximated versions of the WLB and the nonparametric bootstrap, respectively. Then, the input of the resulting generator function is $\boldsymbol{\alpha}$, and the dimension is reduced from n to S. This simple modification dramatically improves the computational efficiency of the GBS. When the sample size n is larger than millions, the vanilla setup considers the generator with the input dimension being millions, but the input dimension for the subgrouped bootstrap is just $S \ (\ll n)$. While this improvement is advantageous, a natural question still remain: does this modification produce a theoretically valid bootstrap distribution? To investigate this theoretical justification, we introduce some helpful notation below.

We distinguish a random variable Y_i and its observed value y_i. Let $Y_1, Y_2, \ldots$ be an iid sequence from the probability measure space $(\Omega, \mathcal{F}, \mathbb{P}_0)$. We denote the empirical probability measure by $\widehat{\mathbb{P}}_n := \sum_{i=1}^{n} \delta_{Y_i}/n$, where δ_x is a point mass measure at $x \in \mathbb{R}$, and let $\mathbb{P}f = \int f d\mathbb{P}$, where $\mathbb{P}$ is a probability measure and f is $\mathbb{P}$-measurable. Let $\hat{\theta}_n(Y_1, \ldots, Y_n) = \mathrm{argmin}_{\theta} \sum_{i=1}^{n} l(\theta; Y_i)$ and denote the true parameter that involves in the true data-generating process by θ_0, and Then, $\sqrt{n}\{\hat{\theta}_n(Y_1, \ldots, Y_n) - \theta_0\}$ can be considered as a function of the empirical process defined as $\sqrt{n}(\mathbb{P}_n - \mathbb{P}_0)$. Suppose that $\sqrt{n}(\mathbb{P}_n - \mathbb{P}_0)$ weakly converges to a probability measure $\mathbb{T}$ defined on some sample space and its sigma field $(\Omega', \mathcal{F}')$. In the regime of bootstrap, what we are interested in is to estimate $\mathbb{T}$ by using some weighted empirical distribution that is $\widehat{\mathbb{P}}_n^* = \sum_{i=1}^{n} w_i \delta_{Y_i}$, where
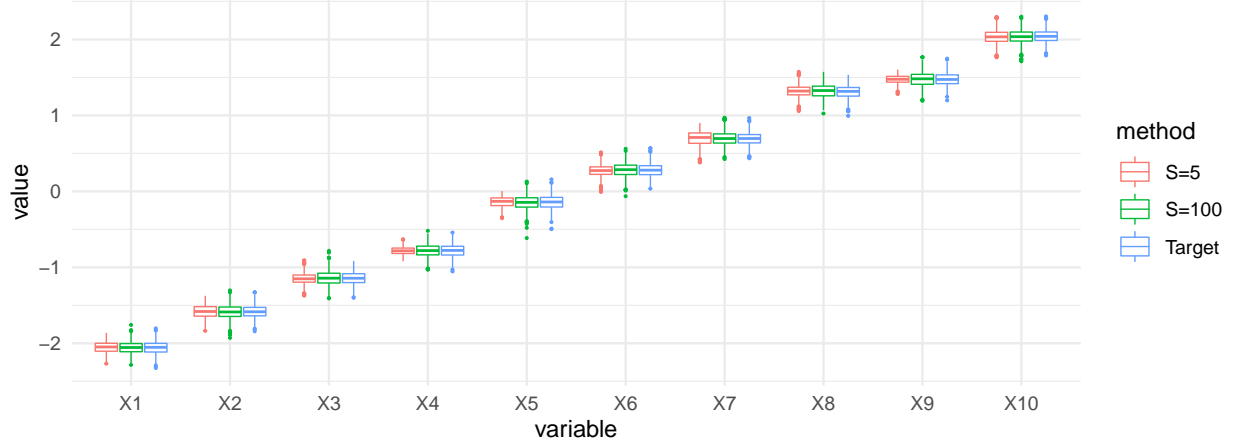
6

Figure 2: Comparisons of block bootstraps across different numbers of blocks.

$w_1, w_2, \ldots$ is an iid weight random variable from a probability measure $\mathbb{P}_{\mathbf{w}}$. In the same sense, the probability measure acts on the block bootstrap is denoted by $\mathbb{P}_{\mathbf{w}_{\alpha}}$. When $\mathbb{P}_{\mathbf{w}}$ is set to be a multinomial probility law with a uniform one parameter, Giné and Zinn (1990) showed that under some measurability condition on a collection of some continuous functions of interest $\mathcal{D}$, the following statement

$$\sqrt{n}(\widehat{\mathbb{P}}_n f - \mathbb{P}_0 f) \to \mathbb{T}f \quad \text{for } f \in \mathcal{D} \quad \text{and} \quad \mathbb{P}_0 f_{\mathcal{D}}^2 < \infty, \tag{4}$$

where $f_{\mathcal{D}}(\omega) = \sup_{f \in \mathcal{D}} |f(\omega)|$ is the envelope function of $\mathcal{D}$, is equivalent to

$$\sqrt{n}(\widehat{\mathbb{P}}_n^* f - \widehat{\mathbb{P}}_n f) \mid Y_1, \ldots, Y_n \to \mathbb{T}f \quad \mathbb{P}_0\text{-a.s.} \tag{5}$$

Præstgaard and Wellner (1993) generalized this result and suggested a set of conditions on $\mathbb{W}$ to guarantee the asymptotic justification in (5). Based on the theoretical findings in Præstgaard and Wellner (1993), the following theorem justifies the use of the proposed subgrouped bootstrap:

Theorem 2.2. *Recall that $\widehat{\theta}_{\mathbf{w}}$ is defined in Theorem 2.1. Suppose that* (4) *holds and* $\{\alpha_1, \ldots, \alpha_S\}^{\mathrm{T}} \sim S \times Dir(1, \ldots, 1)$ *in* (3). *Assume that $S \to \infty$ as $n \to \infty$. Recall that $\widehat{\theta}_{\mathbf{w}}$ is defined in Theorem 2.1. Then,*

$$\sup_{U \in \mathcal{B}} \left| \mathbb{P}_{\mathbf{w}}(\widehat{\theta}_{\mathbf{w}} \in U) - \mathbb{P}_{\boldsymbol{\alpha}}(\widehat{\theta}_{\mathbf{w}_{\boldsymbol{\alpha}}} \in U) \right| \to 0 \quad \text{as } n \to \infty,$$

where $\mathcal{B}$ is the Borel sigma algebra.

Proof. See the appendix. $\square$

Even though the result in Theorem 2.2 is valid in asymptotics, it does not guarantee practically sufficient accuracy of the approximation in finite samples. To examine its finite sample accuracy, we consider a simple example of a linear regression model as
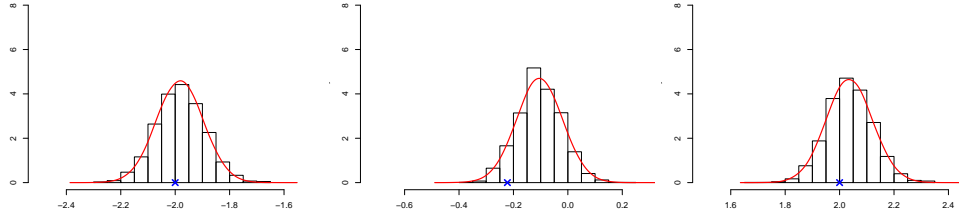
$$y_i = X_i^{\mathrm{T}} \theta + \epsilon_i, \tag{6}$$

where $\epsilon_i \sim N(0, \sigma^2)$ for $i = 1, \ldots, n$. we generate a synthesized data set with $n = 1000$ and $p = 10$, and its true coefficient is a sequence of equi-spaced values between -2 and 2, and $\sigma^2 = 2$.
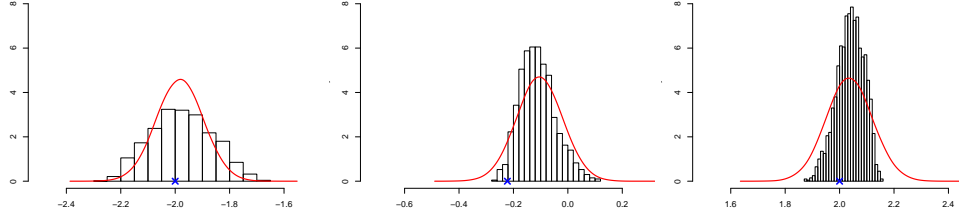
(a) Block bootstrap with with $S = 5$.
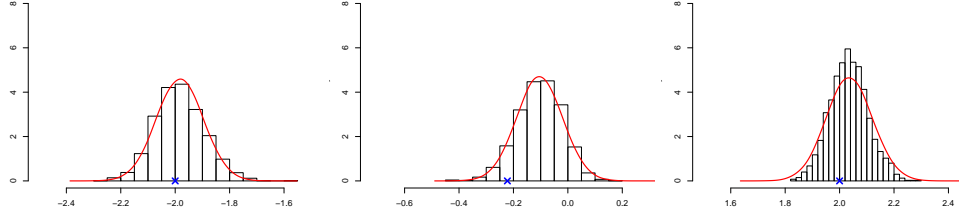
(b) Block bootstrap with with $S = 25$.

(c) Block bootstrap with with $S = 100$.

Figure 3: Histograms of block bootstrap distributions with various S for the coefficient of X_1 (left column), X_5 (middle column), and X_{10} (right column). The red line indicates the density function of the target distribution. Each true coefficient is marked by a blue x.

We generate a data set from this setting and apply the block bootstrap version of the GBS to the synthesized data set.

For the simulated data set, Figure 2 shows boxplots of block bootstrap distributions of MLE with different number of blocks ($S = 5$ and 100). An interesting point in Figrue 2 is that the block bootstrap distribution approximates well the target distribution (the non-block bootstrap distribution), even when the number of subgroups is tiny ($S = 5$). When the number of blocks S is moderately large ($S = 100$), its approximation is indistinguishable from the target. This result is confirmed again in Figure 3, which shows histograms of block bootstrap distributions for a few coefficients. When $S = 5$, while the location of the block bootstrap distribution is close to that of the target one, the scale is unstably different for some coefficients. However, as S increases, the quality of the approximation of the block bootstrap distribution clearly get improved, and when $S = 100$, the target bootstrap distribution is almost perfectly approximated by the block bootstrap distribution. Not only for this toy example, in all simulated and real applications we examined in Section 3 and Section 4, we find that the approximations of the block bootstrap are highly accurate. We use this block bootstrap with $S = 100$ as a default of the GBS in sequel.

2.4 Generator Function Constructed by Neural Network

A popular way to construct a function with a complicated structure is to use neural networks. In applications of machine learning, various results empirically showed that the use of neural networks as a generator of complicated patterns, e.g. image generations, is practically adequate and computationally feasible even for a massive-sized data set (Arjovsky et al., 2017; Goodfellow et al., 2014; Karras et al., 2018; Ledig et al., 2017; Wang et al., 2018). We thus consider a simple form of the network, a modified feed-forwarding neural network, to construct the generator G in (2).

First, we explain a general notion of feed-forwarding neural networks, then we introduce our neural net used in the GBS. Feed-forwarding neural nets are constructed by composing activated linear transformations, and we consider L number of network layers $\{g_1, \ldots, g_L\}$ where $g_l : \mathbb{R}^{N^{(l)}} \mapsto \mathbb{R}^{N^{(l+1)}}$ for $l = 1, \ldots, L$. For the l-th layer, its weight parameter, which is represented by a $N^{(l)} \times N^{(l+1)}$ matrix, and its bias parameter, which is a $N^{(l+1)}$-dimensional vector are denoted by $\mathbf{U} = \{\mathbf{u}_1^{(l)}, \ldots \mathbf{u}_{N^{(l)}}^{(l)}\}$ and $\mathbf{b} = \{b_1^{(l)}, \ldots b_{N^{(l)}}^{(l)}\}$, respectively. Then, the l-th layer g_l is expressible as

$$g_l(\mathbf{Z}) = T(\mathbf{Z}\mathbf{U}^{(l)} + \mathbf{b}^{(l)}) = \left(T(\mathbf{Z}\mathbf{u}_1^{(l)} + b_1^{(l)}), \quad \ldots \quad , T(\mathbf{Z}\mathbf{u}_{N^{(N+1)}}^{(l)} + b_{N^{(N+1)}}^{(l)}) \right).$$

Fitting a neural network requires to choose an activation function $T : \mathbb{R} \mapsto \mathbb{R}$, and commonly used activation functions are a sigmoid function, a hyperbolic tangent function, the Rectified Linear Unit (ReLU), etc. For our generator, we use the ReLU function that is $T(t) = \max\{t, 0\}$ as a default.

A feed-forwarding neural network G is defined by a composition of these layers as

$$G(\mathbf{Z}) = g_L \circ g_{L-1} \circ \cdots \circ g_1(\mathbf{Z}), \tag{7}$$

where g_0 and g_L are the input layer and the output layer, respectively. This means that the structure of the neural network G in (7) is composed in a way that the output of the previous layer will be used as the input of the next layer. The input of the generator G is bootstrap weights $\mathbf{w}$, and these weights are feed-forwarded towards the first hidden-layer g_1, and the output of g_1 is fed to the second hidden-layer g_2, and so on. At the last layer, the bootstrap sample of θ is expressed as a linear combinations of components in the final hidden-layer. We consider a feed-forwarding neural net with three hidden-layers in all simulation and real data studies. The details of the neural net structure are deferred to the appendix.

An issue of this feed-forwarding neural net is that the variation on the weights $\mathbf{w}$ is less likely to be transmitted towards the output as the number of layers increases. As a result, we empirically found that simple feed-forwarding neural nets tend to underestimate the variance of the bootstrap distribution. To overcome this issue, we modify the simple feed-forwarding neural net in a way that every layer receives the bootstrap weight $\mathbf{w}$ as an input. This modification can be formally expressed as a recursive way that follows

$$\begin{aligned} G(\mathbf{w}) &= g_L(\mathbf{Z}_L^*) \\ \mathbf{Z}_{l+1}^* &= \{g_l(\mathbf{Z}_l^*), \mathbf{w}\}, \end{aligned} \tag{8}$$

for $l = 1, \ldots, L - 1$. Figure 4 illustrates this structure of G. At every layer, the bootstrap weight $\mathbf{w}$ is concatenated to the output of the previous layer, and the concatenated results become the
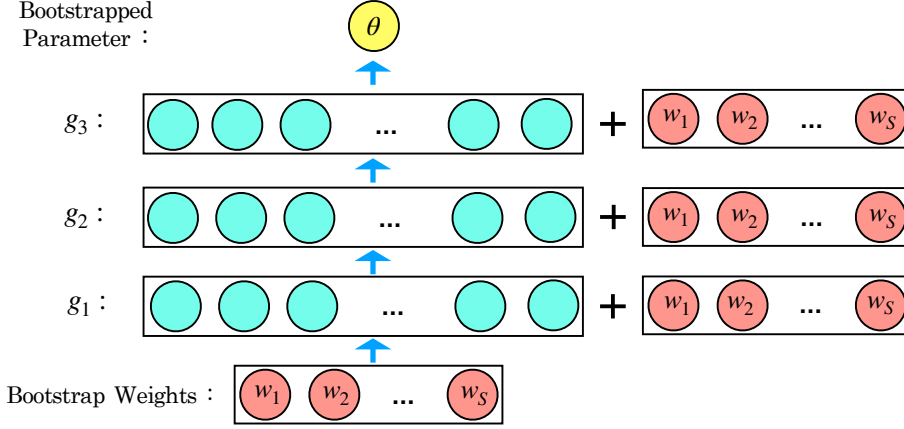
Figure 4: An example of the structure of G via the modified feed-forwarding neural network.

input of the next layer. Compared to the simple feed-forwarding neural network, this modified one directly connects the weight $\mathbf{w}$ and the bootstrap output, and helps gradient flow in training deep neural networks. We use this neural net as a default of the GBS and examine its performance in various examples in Section 3.

2.5 Diagnosis of Optimization Convergence

The GBS procedures enjoy a clear advantage over some other optimization-based UQ procedures in terms of convergence diagnosis in optimization. In the Bayesian paradigm, variational inference and its variants (Blei et al., 2017; Ho et al., 2019; Kingma et al., 2016; Rezende and Mohamed, 2015) are commonly used to approximate the posterior distribution for big data. However, they not only underestimate the posterior variance (MacKay, 2003; Pati et al., 2018; Yao et al., 2018), but also lack diagnostic tools for monitoring optimization convergence. Thus, using the variational Bayes is not ideal for constructing a confidence interval or quantifying uncertainty in statistical inference.

In contrast, the convergence status of the GBS procedures can be diagnosed by comparing the GBS solution and the corresponding classical solution that shares the same weight. Theorem 2.1 shows that when the weight $\mathbf{w}$ is the same, the GBS solution $G(\mathbf{w})$ and the classical solution $\widehat{\theta}_{\mathbf{w}}$ should be equivalent. So, if the optimization procedure for the GBS converges well, the resulting solutions should be close to these from the classical bootstrap. To be more formal, we consider this diagnosis as a hypothesis testing procedure. Since the discrepancy between two results can be interpreted as a paired two sample problem. More specifically, for a small number M, say $M = 5$, we generate $\mathbf{w}_{(1)}, \ldots, \mathbf{w}_{(M)}$ from the pre-specified weight distribution, and we can calculate the discrepancy between results from the GBS and the classical bootstrap procedure. We define the discrepancy as $\mathbf{d}_m = \widehat{G}(\mathbf{w}_{(m)}) - \hat{\theta}_{\mathbf{w}_{(m)}} \in \mathbb{R}^p$. The basic idea is that if $\mathbf{d}_m$ is close enough to zero for $m = 1, \ldots, M$, we can conclude that the optimization procedure is converged well. This procedure is reasonable in a sense that it is rare to match the results from two procedures by chance, when the number of parameters p is moderately large. However, a question "how close towards zero is close enough?" still remains. We thus employ a statistical testing procedure to examine a formal way to diagnose the convergence.

We assume that $d_{m,j} \sim N(\mu_j, \sigma_j^2)$ for $j = 1, \ldots, p$. Then, diagnosing the optimization convergence can be processed via a hypothesis testing such as

$$H_0 : \mu_j = 0 \quad \text{vs.} \quad H_1 : \mu_j \neq 0.$$

This formulation is natural in examining the significance of the discrepancy, but a classical hypothesis testing procedure is mainly designed to detect a significant difference, not to assert no difference. Because the situation where the optimization is converged is corresponding to $H_0 : \mu_j = 0$ for all $j's$, classical significance testing procedures are not appropriate in this problem. Instead, we consider a Bayesian hypothesis testing procedure that provides a measure of evidence that directly compares two hypotheses (or models). For the Bayesian hypothesis testing for each $j = 1, \ldots, p$, under H_1 we set

$$
\begin{aligned}
d_{m,j} \mid \mu_j, \sigma_j^2 &\sim N(\mu_j, \sigma_j^2) \\
\mu_j &\sim N(0, \tau^2 \sigma_j^2) \\
\pi(\sigma_j^2) &\propto 1/\sigma_j^2,
\end{aligned}
\tag{9}
$$

for $m = 1, \ldots, M$. Then, the Bayesian hypothesis testing procedure measures evidence on each hypothesis by comparing marginal likelihoods evaluated under H_0 and H_1, and the odd between these marginal likelihood is called *Bayes factor* (Jeffreys, 1961; Kass and Raftery, 1995). That is,

$$B_{01,j} = \frac{\pi(d_{1,j}, \ldots, d_{M,j} \mid H_0)}{\pi(d_{1,j}, \ldots, d_{M,j} \mid H_1)},$$

where $\pi(\cdot \mid H_0)$ and $\pi(\cdot \mid H_1)$ are the marginal likelihoods under H_0 and H_1, respectively.

By using the conjugacy, we can derive an explicit form of the Bayes factor under (9), and it follows

$$
\log B_{01,j} = \frac{1}{2} \log(M\tau^2 + 1) + \frac{M}{2} \log \left(1 - \frac{(M + 1/\tau^2)\widetilde{d}_j^2}{\sum_{m=1}^M d_{m,j}^2} \right)
\tag{10}
$$

where $\widetilde{d}_j = \sum_{m=1}^M d_{m,j}/(M + 1/\tau^2)$. As noted by Liang et al. (2008) and Johnson and Rossell (2010), the value of Bayes factor is significantly affected by the prior scale parameter τ^2. As the prior on μ_j gets more diffused, more evidence will be assigned on H_0, so when τ^2 is larger, this testing procedure will be more in favor of H_0. We admit that the diagnosis is sensitive to the choice of τ^2, but a simple setting $\tau^2 = 1$ performs well in all examples we tested. Then, our criterion to diagnose the convergence is the sample mean of $\log B_{01,j}$'s; $T_{01} = p^{-1} \sum_{j=1}^p \log B_{01,j}$. As a rule of thumb, we decide that the optimization is converged if $T_{01}^* > r \log(M\tau^2 + 1)/2$ for $0 < r < 1$. We set $r = 0.8$ as a default, and this setting reasonably works well in various examples considered in this paper. We note that this criterion does not follow the standard decision rules suggested by Kass and Raftery (1995). They recommended a decision criterion that a value of $\log B_{01,j}$ larger than 5 indicate "very strong" evidence in favor of H_0, but this rule cannot be applied to our case, since $\log B_{01,j} \leq \log(M\tau^2 + 1)/2$. This means that we may have a situation where the Bayes factor cannot exceed the criterion suggested by Kass and Raftery (1995) when M is small. In algorithm 1, we employ this convergence diagnosis ($T_{01}^* > r \log(M\tau^2 + 1)/2$) within the iterative algorithm, and every 100 iterations we diagnose the optimization chain is converged. If

the optimization procedure consecutively achieves $T_{01}^* > r \log(M\tau^2+1)/2$ five times, we diagnose that the optimization is converged, and we stop the algorithm after N iterations.

This diagnosis procedure of the GBS can be a unique advantage over the other competing procedures, because other optimization-based uncertainty quantification methods are lack of convergence diagnosis tools. For example, while variational inference procedures (Blei et al., 2017; Carbonetto and Stephens, 2012; Pati et al., 2018) are commonly used to approximate the posterior distribution through a family of simple distributions, it is not trivial to check its algorithmic convergence, and practitioners are exposed to a risk of obtaining sub-optimal results. In contrast, the convergence diagnosis of the GBS is straightforward . If the result of the proposed Bayesian test is against the evidence of a convergence, we can tune technical settings in the optimization procedure or run more iterations until the optimization procedure is diagnosed to be converged. This diagnosis procedure requires to evaluate M (a small number less than 10) number of bootstrap samples from the classical procedure, but this extra computational burden would be negligible compared to that required in sampling thousands of classical bootstrap samples.

2.6 Computational Strategy

As examined in various machine learning literature, the optimization for loss functions based on neural networks is highly efficient in computation. That is because neural networks contain simple structure as in (7), and the gradient can be efficiently evaluated by using backpropagation algorithm (Hecht-Nielsen, 1992; Rumelhart et al., 1986) implemented via GPU computing. Once the gradient is evaluated, *Stochastic Gradient Descent* (SGD) algorithm and its variants update the parameter in an iterative sense. The same strategy is used for the computation of GBS. See the appendix for the details of the SGD.

To accelerate the convergence, we use a small number of bootstrap samples evaluated from the conventional bootstrap procedure. We add an extra penalty on the discrepancy between the GBS bootstrap samples and the conventional bootstrap samples, and the resulting loss function follows that

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \, \mathbb{E}_{\mathbf{w}} \left[\sum_{i}^{n} w_i l(G_\phi(\mathbf{w}); y_i) \right] + \zeta \sum_{m=1}^{M} \|G_\phi(\mathbf{w}_{(m)}^*) - \hat{\theta}_{\mathbf{w}_{(m)}^*}\|_2^2, \tag{11}$$

where ϕ is the neural net parameter of the generator G, and ζ is a tuning parameter that controls the influential level of the discrepancy, and $\mathbf{w}_{(m)}^*$ and $\hat{\theta}_{\mathbf{w}_{(m)}^*}$ indicate a pre-sampled weight and its corresponding bootstrap estimator, respectively, for $m = 1, \ldots, M$. By adding this extra penalty term $\zeta \sum_{m=1}^{M} \|G(\mathbf{w}_{(m)}^*) - \hat{\theta}_{\mathbf{w}_{(m)}^*}\|_2^2$, we expect that the convergence speed of the SGD accelerates, and the stability of the optimization is improved. That is because this L_2 distance directly forces the generator to converge towards the target values. However, we cannot overlook a concern of overfitting towards $\hat{\theta}_{\mathbf{w}_{(m)}^*}$'s. If the tuning parameter ζ is large, the generator would be updated in a way that only the generated samples from $\mathbf{w}_{(m)}^*$ for $i = 1, \ldots, M$ are accurate and the other weights would result in sub-optimal generated values. To avoid that, we set $\zeta = 0$ after the optimization status is diagnosed as converged. After converged, we run a long enough length of iterations to prevent the GBS from overfitting. If evaluating a few classical bootstrap samples is not available for some reason, we can ignore the extra penalty term and consider the original loss function in (2). We find that there is a tendency that the convergence of the original form in (2) is relatively slower than the form with the extra penalty term when the data size is massive.

Algorithm 1 An optimization algorithm for the GBS.

- **Preparation step**
- ○ Generate $\mathbf{w}^*_{(m)} \sim H_\mathbf{w}$ (this should be sampled by subgroups; see Section 2.3).
- ○ Evaluate $\hat{\theta}_{\mathbf{w}^*_{(m)}}$ for $m = 1, \ldots, M$.
- **Optimization step**
- ○ Set batch size n_0, ζ, κ, K, K_0, V, T, and N.
- ○ Initialize a $K \times S$ matrix $\boldsymbol{\alpha}$.
- ○ Set $t = 0$ and $A = 0$.

while the stop condition is not satisfied or $t < T$ **do**
-     ○ Let $t = t + 1$.
-     ○ Sub-sample n_0 samples from the original data set, and denote the index set by I_0.
-     ○ Randomly select K_0 number of rows of $\boldsymbol{\alpha}$. For each selected row of $\boldsymbol{\alpha}$, randomly select V number of elements, and replace them with generated values from i.i.d $\exp(1)$.
-     ○ Transform the updated $\boldsymbol{\alpha}$ to a $K \times n$ matrix $\mathbf{w}_{\boldsymbol{\alpha}}$ (see Section 2.3 for details).
-     ○ Calculate $J = \frac{1}{n_0 K} \sum_{k=1}^{K} \sum_{i \in I_0} w^{(k)}_{\boldsymbol{\alpha},i} \frac{\partial}{\partial \phi} l(G_\phi(\mathbf{w}^{(k)}_{\boldsymbol{\alpha}}); y_i)$, where $\mathbf{w}^{(k)}_{\boldsymbol{\alpha}}$, $w^{(k)}_{\boldsymbol{\alpha},i}$ is the k-th row and (k, i)-th element of $\mathbf{w}_{\boldsymbol{\alpha}}$, respectively.
-     (Optional) Set $J = J + \frac{\zeta}{M} \sum_{m=1}^{M} \frac{\partial}{\partial \phi} \| G_\phi(\mathbf{w}^*_{(m)}) - \hat{\theta}_{\mathbf{w}^*_{(m)}} \|_2^2$.
-     ○ Update ϕ with J via a SGD step.
-     **if** $t \mod 100 = 0$ **then**
-         ○ Evaluate $T_{01} = p^{-1} \sum_{j=1}^{p} \log B_{01,j}$, where $\log B_{01,j}$ is defined in (10).
-         **if** $T_{01}^* > r \log(M\tau^2 + 1)$ **then**
-             ○ $A = A + 1$.
-         **else**
-             ○ $A = 0$.
-         **end if**
-     **end if**
-     **if** $A = 5$ **then**
-         ○ Set $\zeta = 0$ in (11) and stop after N iterations.
-     **end if**
end while

Algorithm 1 shows the details of the optimization algorithm for minimizing (2) or (11). The main idea is that the expectation is approximated by a Monte Carlo method with K number of randomly sampled bootstrap weights, and its gradient can be evaluated, then we update the neural net parameter of the generator. If a few bootstrap samples are computable from the conventional bootstrap procedure, we process a convergence diagnosis step at every 100 iteration in the algorithm. If the current optimization status is diagnosed as converged five time in a row via the diagnosis procedure introduced in Section 2.5, we set $\zeta = 0$ and run N iterations more, then the algorithm stops.

3 Examples and Simulation Studies

3.1 Gaussian Linear Regression

We consider an example of linear regression models in (6) to examine the performance of the GBS in evaluating the bootstrap distribution. We note that the exact form of the sampling distribution of MLE is available for linear models, and the advantage of using bootstrapping is minimal for this simple example in practice. Nerverthless, examining linear models is still meaningful in a sense that the GBS is a general procedure that is applicable to a wide range of models even under

	Correlated						Independent					
$p=50$	$n=500$			$n=1000$			$n=500$			$n=1000$		
Method	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)
GBS	0.969	0.432	37.7+0.1	0.936	0.205	45.2+0.1	0.936	0.110	39.4+0.1	0.957	0.052	47.9+0.1
BT(1C)	0.932	0.435	16.2	0.933	0.206	25.1	0.931	0.113	18.4	0.937	0.052	27.3
BT(25C)			0.9			1.3			0.9			1.3
Target	0.951	0.429		0.950	0.204		0.953	0.109		0.954	0.052	
$p=500$	$n=3000$			$n=10000$			$n=3000$			$n=10000$		
Method	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)
GBS	0.928	0.805	61.4+0.1	0.923	0.212	112.8+0.1	0.915	0.205	63.8+0.1	0.928	0.053	114.9+0.1
BT(1C)	0.910	0.805	1924.3	0.932	0.211	11209.8	0.911	0.202	1967.4	0.932	0.053	11024.2
BT(25C)			92.6			525.7			91.1			520.2
Target	0.948	0.804		0.948	0.211		0.948	0.202		0.948	0.053	
$p=2000$	$n=20000$			$n=50000$			$n=20000$			$n=50000$		
Method	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)
GBS	0.940	0.448	244.9+0.1	0.966	0.169	366.3+0.1	0.914	0.113	245.7+0.1	0.948	0.044	361.2+0.1
BT(1C)	NA	NA	> 1 week	NA	NA	> 2 weeks	NA	NA	> 1 week	NA	NA	> 2 weeks
BT(25C)	NA	NA	> 8 hours	NA	NA	> 14 hours	NA	NA	> 8 hours	NA	NA	> 14 hours
Target	0.949	0.446		0.949	0.167		0.949	0.112		0.950	0.041	

Table 1: Linear regression results of the coverage of 95% confidence intervals, MSE, and the actual computation time in second to generate 10,000 bootstrap samples for each procedure. We omit "Cov" and "MSE" of bootstrap(25C), because they are the same with those of bootstrap(1C).

situations where the evaluation of the sampling distribution is nontrivial.

To more rigorously examine the performance of the GBS, we consider a simulation study. The settings of the simulation follows $n \in \{500, 1000, 3000, 10000, 20000, 50000\}$ and $p \in \{50, 500, 2000\}$, $\sigma^2 = 1$, and the true regression coefficient are equi-spaced between -2 and 2. For simplicity, we assume that σ^2 is known. Each covariate X_i is i) independently generated from standard Gaussian or ii) dependent such as $X_i \sim N(0, \Sigma)$, where $\Sigma_{jk} = 1$, if $j = k$, and 0.5, if $j \neq k$, and they are denoted by "Independent" and "Correlated" in Table 1, respectively. For each bootstrap procedures, we generate 10,000 bootstrap estimators. We report the average of the results across 100 independently replicated data sets.

This simulation study is implemented on a workstation equipped with a `AMD Threadripper 2990WX` CPU with 32 cores and 64 threads of base clock 3.0GHz, two `Nvidia RTX2080Ti` GPUs, and 128GB memory with DDR4 2,800Mhz clock speed. To implement the WLB, we use `lm` and `glm` functions in R. For the GBS, `Pytorch`, which is a GPU-efficient deep learning library in `Python`, is used to optimize the generator function G. A parallel computing environment with 25 cores is considered for the standard bootstrap procedures to reduce the computation time via an R package `snowfall`. For the GBS procedures, we separately report the training time to train $\widehat{G}$ and the time to generate samples from the trained generator. For example of the GBS, on the first row of Table 1, "37.7+0.1" means that it takes 37.7 seconds to optimize the generator function and 0.1 seconds to generate 10,000 bootstrap samples after the training. For large-sized data sets with $p = 2000$, because the classical bootstraps are computationally infeasible, we mark their result by "NA", and we estimate their computation time from the time in generating 20 bootstrap samples, instead of 10000. We also consider the target distribution of the bootstrap distribution, which is $N(\widehat{\theta}_{MLE}, \sigma^2(X^{\mathrm{T}}X)^{-1})$, to examine the approximation performance of the GBS. All MSE and the coverage results of the classical bootstrap are reported from the parallel environment with 25 cores.

For linear models, the explicit form of the solution of (1) is available, i.e. $\widehat{\theta}_{\mathbf{w}} = (X^{\mathrm{T}}D_{\mathbf{w}}X)^{-1}X^{\mathrm{T}}D_{\mathbf{w}}\mathbf{y}$,

where $D_{\mathbf{w}}$ is a diagonal matrix of $\mathbf{w}$. This means that bootstrap sampling can be computationally more convenient and faster than other arbitrary models. However, despite this computational efficiency, the results in Table 1 suggest that implementing a classical bootstrap for a large-sized data set, like $(n, p) = (50000, 2000)$, is practically challenging and requires a huge amount of computing resource. With a single-threading computation, it takes more than 2 weeks to generate 10,000 bootstrap samples, and even under a parallel computing environment with 25 cores, the computation is expected to be longer than 14 hours. In contrast, the training time of the GBS procedure is around 6 minutes under the same setting, and the generation time for bootstrap samples is expected to 0.1 second in all scenarios. This result shows that the GBS is capable of accelerating computing bootstrap distribution by hundreds of folds in the linear regression examples where conventional bootstrap procedures are favored in computation. Moreover, the 95% coverage of the confidence intervals constructed by the GBS is stably close to that of the WLB where the GBS is approximating.

3.2   Robust regression model

When outliers exist, it is reasonable to consider a robust regression model based on an *M-estimator* that minimizes the following cost function:

$$\sum_{i=1}^{n} \psi(y_i - X_i^{\mathrm{T}}\theta), \tag{12}$$

where ψ is a residual function that satisfies *i)* $\psi(t) > 0$ for all $t \in \mathbb{R}$; *ii)* $\psi(0) = 0$; *iii)* $\psi(t) = \psi(-t)$; *iv)* ψ is monotonically increasing (Rousseeuw and Leroy, 2005). Common choices of ψ include the *Huber* function, the absolute value function for median regression, the *bisquare* function, etc. Unlike the Gaussian linear regression model previously examined, this M-estimator does not have a closed form of the solution, and its computation is commonly conducted by a computationally expensive procedure like the *Iteratively Reweighted Least Square* (IRLS, Street et al. (1988)). This computational burden hurdles the use of bootstrapping procedures to quantify the uncertainty in the statistical procedure. Moreover, it is also challenging to derive the sampling distribution of θ in these robust regression models. That is because their sampling distribution depends on the true distribution of errors, which is unknown, and estimating the error distribution is extremely challenging in general. Thus, the UQ on the statistical inference based on the M-estimator is practically difficult to evaluate.

We apply the GBS to overcome these bottlenecks in UQ, and provide a scalable way to generate a bootstrap distribution of the M-estimator. By directly applying the form (2), the objective function of the GBS follows:

$$\min_{G} \mathbb{E}_{\mathbf{w}} \left[\sum_{i=1}^{n} w_i \psi(y_i - X_i^{\mathrm{T}} G(\mathbf{w})) \right]. \tag{13}$$

We examine the accuracy of the GBS in computing the bootstrap distribution by using some simulated data set that are generated from the same setting used for linear regression models, except that the errors are generated from t distribution with degree of freedom 3. We use an R package `robust` to implement the WLB as a reference of the GBS.

15

	Correlated						Independent					
$p=100$		$n=2000$			$n=5000$			$n=2000$			$n=5000$	
Method	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)
GBS	0.951	0.281	208.6+0.1	0.966	0.117	243.0+0.1	0.964	0.069	265.9+0.1	0.960	0.028	295.6+0.1
BT(1C)	0.98	0.268	3006.0	0.976	0.096	15772.0	0.985	0.065	3102.8	0.979	0.024	14610.9
BT(25C)			144.5			756.7			146.8			684.4
BT(GPU)			> 3 hours			> 3 hours			> 3 hours			> 3 hours
$p=300$		$n=10000$			$n=20000$			$n=10000$			$n=20000$	
Method	Cov	MSE	Time	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)
GBS	0.987	0.179	214.9+0.1	0.981	0.087	351.3+0.1	0.937	0.042	342.6	0.940	0.023	478.2
BT(1C)	NA	NA	> 1 week	NA	NA	> 2 weeks	NA	NA	> 1 week	NA	NA	> 2 weeks
BT(25C)	NA	NA	> 7 hours	NA	NA	> 20 hours	NA	NA	> 7 hours	NA	NA	> 20 hours
BT(GPU)	NA	NA	> 3 hours	NA	NA	> 3 hours	NA	NA	> 3 hours	NA	NA	> 3 hours

Table 2: Results for least absolute deviation regression.

3.3 Logistic regression model

Suppose we have observations (y_i, X_i), $i = 1, \ldots, n$, where the y_i's are binary, equaling either 0 or 1. Consider the standard logistic regression model, in which the log odds of the i-th event $y_i = 1$ is a linear function of its covariate X_i, i.e.,

$$y_i \sim Bernoulli(\mu_i), \text{ where } \mu_i = \frac{1}{1 + \exp\{-X_i^{\mathrm{T}}\theta\}}, \ \theta = (\theta_1, \ldots, \theta_p)^T.$$

A standard way to construct a confidence interval for a regression coefficient θ_j is based on a Wald-type procedure. However, a Wald-type CI highly relies on the asymptotic normality of the MLE, it can perform poorly if the sample size is small or moderate. In the generalized linear model, a Wald-type CI can also perform poorly when the distribution of the parameter estimator is skewed or if the standard error is a poor estimate of the standard deviation of the estimator. For the logistic regression model, one of the most widely used generalized linear models, profile likelihood confidence interval can provide better coverage (Royston,2007). Profile likelihood confidence interval is derived from the asymptotic χ^2 distribution of the likelihood-ratio test statistic. The confidence interval for a single entry of a vector is constructed by repeatedly maximizing over the other parameters. Since a grid of potential values of this entry are evaluated in this process, profile likelihood CI can be time and computation consuming when the dimension of the parameter is high. Another alternative to the Wald-type CI is the bootstrap confidence interval. Bootstrap CI does not involve any assumptions of the sampling distribution of the estimate but it can be inefficient when the model is complicated. In this section, we compared the proposed GBS CI with the profile likelihood CI and the bootstrap CI from the perspective of 95% coverage rate, MSE and computation time.

To generate simulated data sets, we assume the true regression coefficients θ are equi-spaced numbers between -0.5 and 0.5. We follow the same setting, used for the lineare regression examples, in generating simulated covariates. Analogous to the setting in linear regression, we compared the performance of GBS to the standard bootstrap and the profile likelihood confidence interval when covariates are correlated and independent, with sample size $n \in \{500, 5000, 10000\}$ and dimension of covariates $p \in \{10, 50\}$. For each configuration, 100 replicates were conducted in evaluating MSE, the coverage of the CI, and actual computation time. The results are demonstrated in Table 3.

16

	Correlated						Independent					
$p=10$	$n=500$			$n=5000$			$n=500$			$n=5000$		
Method	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)
GBS	0.922	0.336	121.4+0.1	0.919	0.033	127.4+0.1	0.895	0.121	116.4+0.1	0.929	0.010	124.8+0.1
BT(1C)	0.939	0.339	25.1	0.945	0.033	134.5	0.943	0.123	24.3	0.953	0.010	129.3
BT(25C)			6.8			14.7			6.7			14.2
BT(GPU)			> 12 hours			> 12 hours			> 12 hours			> 12 hours
ProfileLik	0.949	0.322	0.2	0.955	0.033	0.6	0.957	0.114	0.2	0.955	0.010	0.6
$p=100$	$n=5000$			$n=10000$			$n=5000$			$n=10000$		
Method	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)
GBS	0.942	0.530	246.5+0.1	0.953	0.239	280.3+0.1	0.928	0.255	236.4+0.1	0.944	0.105	267.8+0.1
BT(1C)	0.933	0.525	1284.2	0.937	0.240	3283.7	0.911	0.245	1387.9	0.926	0.104	3236.2
BT(25C)			60.5			148.9			66.2			147.5
BT(GPU)			> 15 hours			> 15 hours			> 15 hours			> 15 hours
ProfileLik	0.947	0.488	55.3	0.947	0.231	152.8	0.939	0.205	51.9	0.945	0.095	150.9
$p=400$	$n=20000$			$n=50000$			$n=20000$			$n=50000$		
Method	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)	Cov	MSE	Time(sec)
GBS	0.952	0.938	607.5+0.1	0.923	0.359	683.9+0.1	0.913	0.625	571.8+0.1	0.971	0.275	655.4+0.1
BT(1C)	NA	NA	> 2 days	NA	NA	> 4 days	NA	NA	> 2 days	NA	NA	> 4 days
BT(25C)	NA	NA	> 2 hours	NA	NA	> 5 hours	NA	NA	> 2 hours	NA	NA	> 5 hours
BT(GPU)	NA	NA	> 3 days	NA	NA	> 1 week	NA	NA	> 3 days	NA	NA	> 1 week
ProfileLik	NA	NA	> 5 hours	NA	NA	> 13 hours	NA	NA	> 5 hours	NA	NA	> 13 hours

Table 3: Results for logistic regression models.

3.4 Quantile Regression Model

Quantile regression analysis has been commonly used in modeling a relation between a certain quantile of the response and covariates, and it models a conditional quantile of the response by a linear transform of the covariate as follows: for a given $\tau \in (0, 1)$,

$$Q_Y(\tau; X_i) = X_i^{\mathrm{T}}\theta, \tag{14}$$

where $Q_Y(\tau; X_i)$ is the conditional τ-th quantile of the response given X_i. Bootstrap samples of an estimate of θ can be obtained by setting the loss function in (1) as

$$l(\theta; y_i) := \rho_\tau(y_i - X_i^{\mathrm{T}}\theta), \tag{15}$$

where $\rho_\tau(t) = t(\tau - I(t < 0))$. Unlike simple parametric models such as logistic and linear regression models, it is challenging to derive its asymptotic sampling distribution of the coefficient estimator in quantile regression. As a result, uncertainty quantification procedures, such as estimations of standard errors, variance-covariance matrices, and confidence interval, cannot be conducted from theoretical results. That is because direct estimation of the asymptotic sampling distribution requires an estimate of the regression error density function, and this density function estimation itself is an extremely difficult task under high-dimensional settings (Koenker, 1994). Instead, in routine applications of quantile regression analysis, bootstrap procedures are popular to approximate the behavior of the sampling distribution (Feng et al., 2011; Hahn, 1995; Kocherginsky et al., 2005). However, the nature of resampling in bootstrap procedures imposes heavy burden in computation. When a practitioner is interested in investigating multiple quantile levels, it is also required to evaluate multiple bootstrap distributions at different quantile levels. These computational repetitions are disastrous and practically infeasible when the size of data is massive.

To relieve this computational bottleneck, we shall apply our GBS to bootstrapping quantile regression models by plugging $l(y_i; G(\mathbf{w}, \tau)) = \rho_\tau(y_i - X_i^{\mathrm{T}}G(\mathbf{w}, \tau))$ in (2). We note that we

make a minor modification from the original loss function (2) to account for varying τ. The main idea of the modification is that we can consider each optimizer of the loss function as a function of $\mathbf{w}$ and τ. This point is reflected in the resulting loss function below:

$$\widehat{G} = \underset{G}{\text{argmin}} \, \mathbb{E}_{\mathbf{w},\tau} \left[\sum_i^n w_i \rho_\tau(y_i - X_i^{\mathrm{T}} G(\mathbf{w}, \tau)) \right], \tag{16}$$

where $\mathbb{E}_{\mathbf{w},\tau}$ is the expectation operator on $\mathbf{w}$ and τ, assuming that τ follows some distribution F_τ whose support is (0,1) and independent with $\mathbf{w}$. A default choice is that $F_\tau = \text{Unif}(0,1)$ and $\mathbf{w}$ independently follows a scaled Dirichlet distribution with unit parameter. Like the main idea of the GBS, this loss function (16) for the quantile regression views the optimizer of (1) as a function of weights $\mathbf{w}$ and a quantile level τ. As a corollary of Theorem 2.1, one can show that a well-trained generator $\widehat{G}$ in (16) generates the same bootstrap samples with those produced from standard bootstrap procedures in (1) under the same weights:

Corollary 3.1. *Suppose that $\widehat{G}$ is the solution of (16) and $\widehat{\theta}_{\mathbf{w},\tau} = \text{argmin}_\theta \sum_i^n w_i \rho_\tau(y_i - X_i^{\mathrm{T}}\theta)$ for a given $\mathbf{w}$ and $\tau \in \mathbb{R}$. Then, $\widehat{G}(\mathbf{w}, \tau) = \widehat{\theta}_{\mathbf{w},\tau}$.*

Proof. This is an immediate corollary of Theorem 2.1. □

This corollary states that once the generator is trained, we can easily produce bootstrap samples of the quantile regression under a specific quantile level by just plugging in randomly generated $\mathbf{w}$'s and the quantile level τ of interest into the generator. This means that the GBS procedure does not require any re-optimizations for a large number of different weights, which is usually computationally intensive when examining multiple different quantile levels. As a consequence, it is expected that the computing time of the bootstrapping would be significantly reduced without compromising the computational accuracy.

To examine a practical usefulness of this procedure, we examine an example considered in Feng et al. (2011), and a simulated data set is generated from a model with $n = 500$ and $p = 5$, such as $y_i = X_i^{\mathrm{T}}\theta_0 + 3^{-1/2}[2 + \{1 + (X_{1i} - 8)^2 + X_{2i}\}/10]\epsilon_i$, where $\theta_0 = \{1,1,1,1,1\}^{\mathrm{T}}$ and $\epsilon_i \sim t_3$. We choose X_{2i} to be one for the first 80% of the observations and zero for the rest, and the other covariates are independently sampled from the standard log-normal distribution.

Figure 5 illustrates 95% confidence bands computed via the GBS and the standard bootstrap (WLB) over quantiles varying from 0.1 and 0.9. Not only the results show that our GBS generates almost identical bootstrap distribution with that of the standard bootstrap, but both 95% confidence bands also successfully cover the true coefficient. We note that without an repetitive computation, this results are obtained from just substituting different values of weights and a value of tuning parameter. In the next section, we further extend the idea of the GBS to other statistical procedures with repetitive computations, and we show that the GBS improve their computational efficiency.

3.5 Gaussian Mixture Model

We consider a Gaussian mixture model with K components as

$$y_i \sim N(\mu_k, \Sigma_k) \quad \text{with probability } \pi_k, \tag{17}$$

where μ_k and Σ_k are mean and variance of the k-th Gaussian component, respectively, for $k = 1, \ldots, K$. This Gaussian mixture model can be modeled by a density function such as $\prod_{i=1}^n \{\pi_k \phi(y_i; \mu_k, \Sigma_k)\}^{\gamma_{ik}}$,
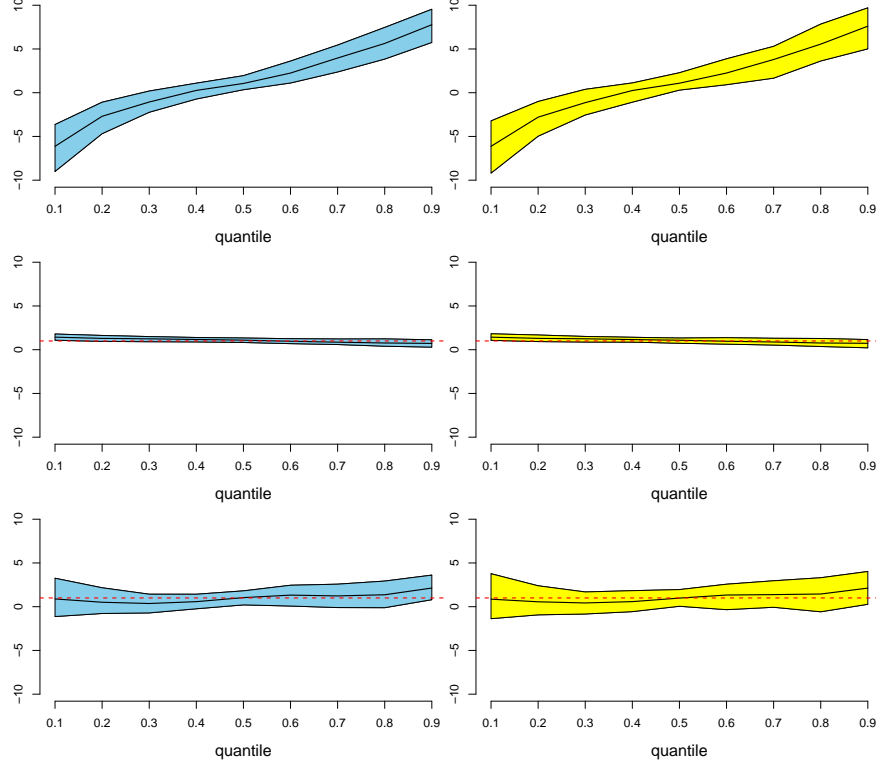
Figure 5: Comparisons between 95% confidence bands from the GBS (left column) and the classical bootstrap (right column) across quantile. The top figure is for the coefficient of the intercept; the middle is for the coefficient of X_1; the bottom is for the coefficient of X_2. The true coefficients are marked by red dotted lines.

where ϕ is a Gaussian density function and $\sum_{k=1}^{K} \gamma_{ik} = 1$ and $\sum_{k=1}^{K} \pi_k = 1$ for $i = 1, \ldots, n$. Thus, we can construct the objective function of a bootstrap procedure for Gaussian mixture models as

$$\{\widehat{\gamma}_{ki}^{(b)}, \widehat{\pi}_k \widehat{\mu}_k^{(b)}, \widehat{\Sigma}_k^{(b)}\}_{i,k} = \underset{\gamma_{ik}, \pi_k, \mu_k, \Sigma_k}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{k=1}^{K} w_i^{(b)} \gamma_{ik} \log\{\pi_k \phi(y_i; \mu_k, \Sigma_k)\}.$$

Since we can interpret each $\{\widehat{\pi}_{ki}^{(b)}, \widehat{\mu}_k^{(b)}, \widehat{\Sigma}_k^{(b)}\}_{i,k}$ as an output of a generator function from an input value $\mathbf{w}^{(b)}$, the GBS idea can be applied to this Gaussian mixture model by following as

$$\begin{aligned}
&\{\widehat{G}_\pi, \widehat{G}_\gamma, \widehat{G}_\mu, \widehat{G}_\Sigma\} \\
&= \underset{G_\gamma, G_\pi, G_\mu, G_\Sigma}{\operatorname{argmax}} \mathbb{E}_{\mathbf{w}} \left(\sum_{i=1}^{n} \sum_{k=1}^{K} w_i G_{\gamma,k}(y_i, \mathbf{w}) \log[G_{\pi,k}(\mathbf{w}) \phi(y_i; G_{\mu,k}(\mathbf{w})_k, \{G_{\Sigma,k}(\mathbf{w})^{\mathrm{T}} G_{\Sigma,k}(\mathbf{w})\}^{-1})] \right),
\end{aligned}$$

where $G_{\gamma,k}$, $G_{\pi,k}$, $G_{\mu,k}$, and $G_{\Sigma,k}$ are the parameters of the k-th Gaussian component. Also, the generator functions are mapping as $G_\gamma : \mathbb{R}^{p+S} \mapsto \Delta_K$, $G_\pi : \mathbb{R}^S \mapsto \Delta_K$, $G_\mu : \mathbb{R}^S \mapsto \mathbb{R}^p \times \{1, \ldots, k\}$, and $G_\Sigma : \mathbb{R}^S \mapsto \Lambda^p \times \{1, \ldots, k\}$, where Δ_K is the space of K-dimensional simplex, i.e. $\Delta_K = \{(\pi_1, \ldots, \pi_K) : \sum_{k=1}^{K} \pi_k = 1, \quad \pi_k \geq 0\}$, and Λ^p is the space of lower triangular matrices so that for any $A \in \Lambda^p$, $A^{\mathrm{T}} A$ is always positive definite. We parameterize the
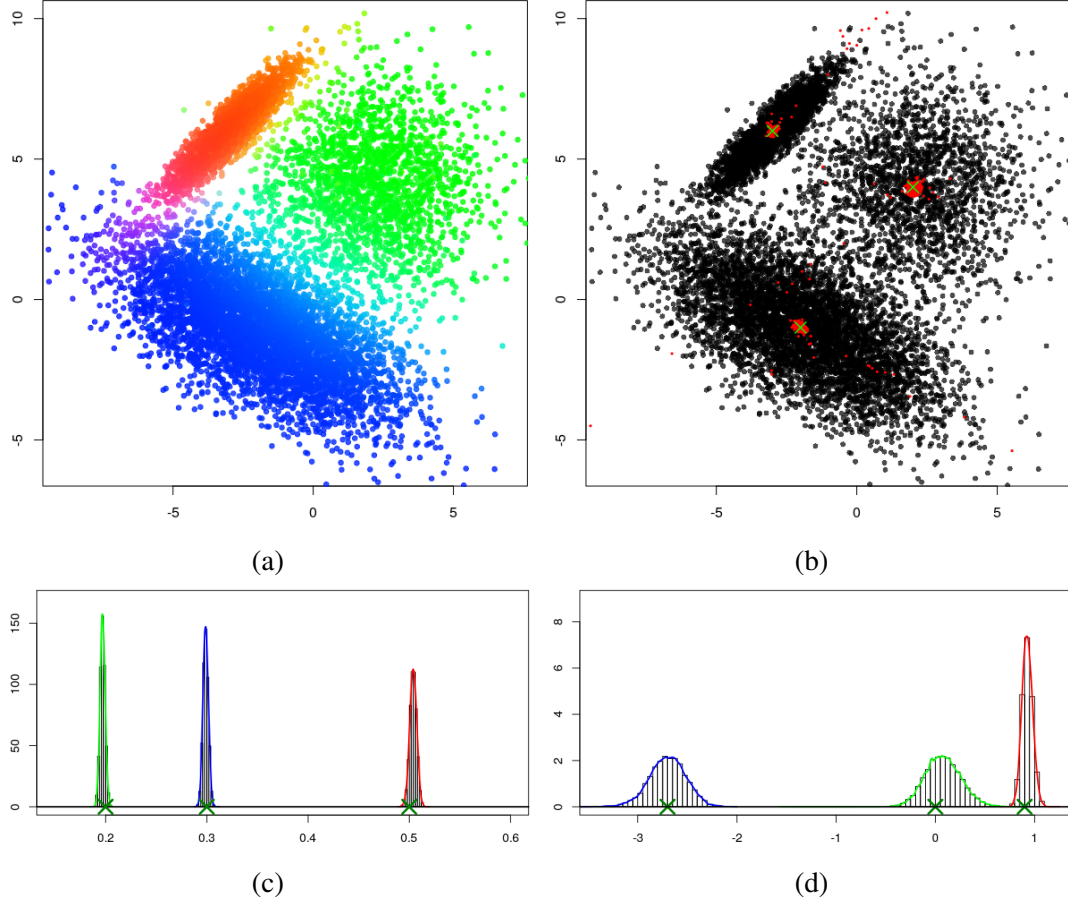
19

Figure 6: An application of the GBS to a Gaussian mixture model. (a) estimated individual component proportion from the bootstrap distribution is illustrated as a mixture of red, green, and blue colors; (b) The bootstrap distribution of location parameters. The 100,000 bootstrap samples of the location parameters are marked by red dots, and the true locations are marked by X. (c) A bootstrap distribution of component proportion parameters. The colors (red, blue, and green) are matched with these in (a) for each cluster. (d) A bootstrap distribution of the off-diagonal term in the covariance matrix for each Gaussian component, and the true values are noted by X.

inverse of the covariance matrix as the product of the lower triangular matrix to avoid a matrix inversion in the likelihood.

We consider a synthetic example of a 2-dimensional Gaussian mixture model whose sample size is moderate, but large enough to impede the use of boostrapping procedures due to computational burdens ($n = 10000$).

$$(\pi_1, \pi_2, \pi_3) = (0.2, 0.5, 0.3), \quad \mu_1 = (2, 4)^{\mathrm{T}}, \quad \mu_2 = (-2, -1)^{\mathrm{T}}, \quad \mu_3 = (-3, -6)^{\mathrm{T}},$$
$$\Sigma_1 = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 6 & -2.7 \\ -2.7 & 3 \end{bmatrix}, \quad \Sigma_3 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}.$$

The bootstrap distribution of this simulation setting is illustrated in Figure 6. The figure shows that the GBS reasonably induces bootstrap distributions as the bootstrap samples of μ_1, μ_2, and μ_3 in Figure 6 (b) cover the true parameter well. Figure 6 (c) and (d) also present the bootstrap distribu-

Model	Linear	LAD	Logistic	GaussMix
(n, p)	$(50000, 2000)$	$(20000, 300)$	$(50000, 400)$	$(10000, 2)$
GBS	about 6 min	about 6 min	about 11 min	about 15 min
BT(1C)	> 2 weeks	> 2 weeks	> 4 days	> 44 days
BT(25C)	> 14 hours	> 20 hours	> 5 hours	> 2 days

Table 4: A brief summary of the comparisons of computation time in generating 10,000 bootstrap samples.

tion distribution of γ_k and $\Sigma_{22,k}$ for $k = 1, 2, 3$, respectively. Because the classical bootstrap takes 44 days to evaluate 10000 bootstrap samples, we cannot compare the results from the GBS with that of the classical bootstrap. However, we note that the computation time of the GBS takes only 15 minutes to get 10000 bootstrap sample. This computational acceleration is not ignorable.

Finally, we make a summary of the computation times of the GBS and the classical counterpart for the considered models in Table 4. This results shows that the GBS improves the computational efficiency at a fold of hundred, compared with a parallel computing with 25 cores. Compared with a single core computing, the speed-up in computation is almost a fold of thousands.

4   Some Extensions of Generative Bootstrap Sampler

4.1   Tuning Parameter Selection

In practice, tuning parameter selection has been a thorny issue and computationally expensive for many machine learning algorithms. A main reason is that many candidates of the tuning parameters need to be considered and for each candidate the same computational (optimization) procedure has to be repeated and an evaluation score (such as the cross-validation error or the value of an information criterion) of the resulting prediction model is computed. However, when the data size is massive, the required repetitive computations can be practically infeasible. To overcome this issue, we apply the GBS in computing the MLEs for different tuning parameters, and the results show that our GBS dramatically improves computational efficiency by bypassing undesirable repetitions in computation.

Consider a model with a loss function $\sum_{i=1}^{n} l_\lambda(\theta; y_i)$ for a tuning parameter $\lambda \in \Lambda \subset \mathbb{R}^q$, and a popular example is a penalized likelihood via setting $l_\lambda(y_i; \theta) = -\log L(y_i \mid \theta) + \lambda f(\theta)/n$, where L is a likelihood function and f is a penalty function on θ. These forms are common and some useful examples include Gaussian process regression (Kirk and Stumpf, 2009; Rasmussen and Williams, 2006) and splines (Wahba, 1978, 1990), LASSO (Tibshirani, 1996), group LASSO (Yuan and Lin, 2006), SCAD (Fan and Li, 2001), MCP (Zhang, 2010), etc. In a framework of the GBS, the tuning parameter can be viewed as an extra input of the generator function $G : \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^p$, and this point can be formalized as:

$$\widehat{G} = \operatorname*{argmin}_{G} \mathbb{E}_{\mathbf{w}, \lambda} \left[\sum_{i=1}^{n} w_i l_\lambda(G(\mathbf{w}, \lambda); y_i) \right], \tag{18}$$

where $\mathbb{E}_{\mathbf{w}, \lambda}$ is the expectation operator with respect to $\mathbf{w}$ and λ, and λ is assumed to follow a pre-specified distribution that attains a large enough amount of density on interesting regions in the tuning parameter space. It is also reasonable to assume that $\mathbf{w}$ and λ are independent. When
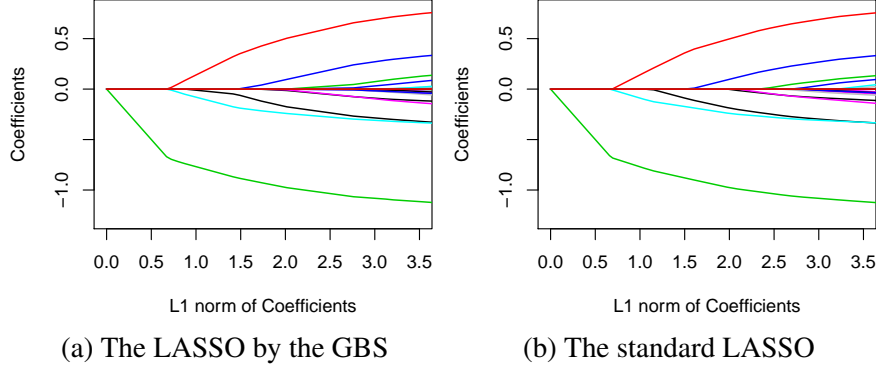
(a) The LASSO by the GBS   (b) The standard LASSO

Figure 7: Solution paths of a linear regression example.

a practitioner is not interested in constructing a bootstrapping distribution and only wants to focus on tuning parameter evaluation, he or she can just simply set $w_i = 1$ for $i = 1, \ldots, n$.

LASSO. A popular example of penalized likelihood procedures is the LASSO, and the corresponding loss function for the GBS version is expressible as

$$\mathbb{E}_{\mathbf{w}, \lambda} \left[\sum_{i=1}^{n} w_i \{y_i - X_i^{\mathrm{T}} G(\mathbf{w}, \lambda)\}^2 + \lambda \|G(\mathbf{w}, \lambda)\|_1 \right], \tag{19}$$

where $\| \cdot \|_1$ is the L_1 norm. By changing the log-likelihood part and the penalty part in (19), we can easily apply the GBS to other penalized likelihood procedures.

Like the vanilla GBS in the previous section, after finding the optimal solution $\widehat{G}$, a bootstrapped sample from $\mathbf{w}^*$ and λ^*, which minimizes $\sum_{i=1}^{n} w_i^* l(\theta; y_i) + \lambda^* f(\theta)$ with respect to θ, is equivalent to $\widehat{G}(\mathbf{w}^*, \lambda^*)$. By changing the value of λ^*, the trained generator function $\widehat{G}$ provides the corresponding bootstrap distribution.

The usefulness of this new procedure is well-illustrated in Figure 7 for the GBS application to the LASSO. A linear regression example is examined here again and a similar setting with the previous example is considered. A sparsity condition is assumed on the true regression coefficients $\beta_0 = \{-0.5, 1, -1.5, 0.5, -0.5, 0, \ldots, 0\}$ with $n = 500$ and $p = 50$. Figure 7 shows solution-path plots that depict the relations between the tuning parameter choices and the corresponding estimated LASSO estimators. The x-axis indicates the L_1 norm of the LASSO estimators based on a series of λs and the y-axis is the estimated values of the LASSO estimators, so each colored line illustrates the behavior of the shrinkage estimator for each coefficient as λ varies. After the generator is trained by minimizing (19), Figure 7 (a) illustrates the mean of bootstrap distribution of each coefficient by simply substituting λ, which varies from 1.0 to 170.0, into $\widehat{G}$. The resulting solution-path of the GBS-LASSO procedure shows that the proposed method successfully approximates the standard LASSO solution-path, and this result is evaluated without optimizing multiple LASSO problems for different tuning parameters. As a result, it is expected that the GBS significantly reduces the computational burden in tuning parameter selection when the data set is massive.
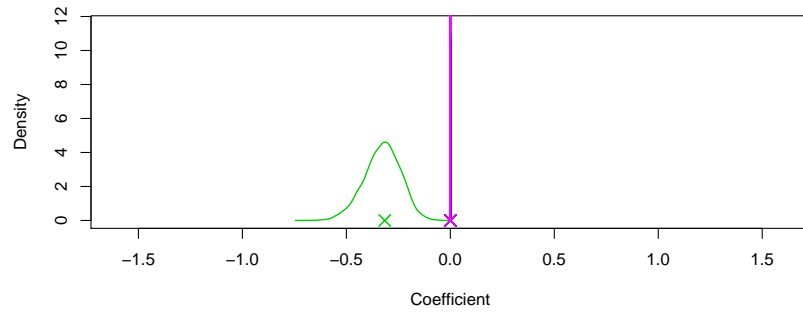
To get more insight on the bootstrapped GBS LASSO procedure, it would be helpful to investigate how its bootstrapped distribution dynamically shrinks as λ varies. Figure 8 illustrates the
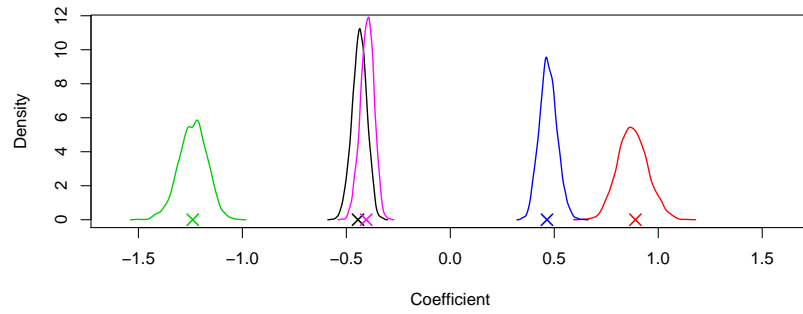
(a) $\lambda = 2.77$
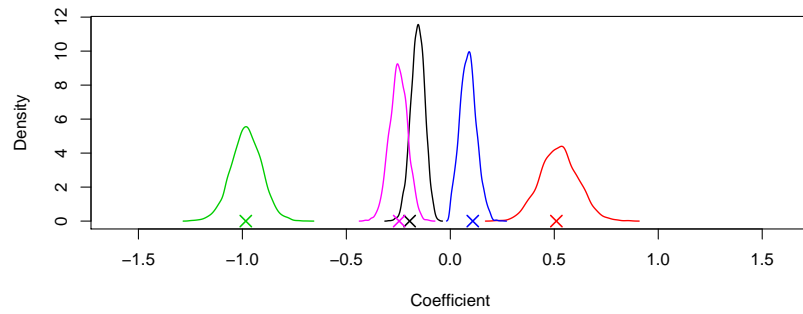
(b) $\lambda = 37.12$

(c) $\lambda = 167.36$

Figure 8: An example of the GBS bootstrapped distributions of LASSO for various tuning parameter λ. Each "X" mark indicates the standard LASSO estimator corresponding to the same colored bootstrap distribution.

evaluated bootstrapped GBS distribution of LASSO when $\lambda \in \{2.77, 37.12, 167.36\}$. The results not only show that the bootstrapped distribution of each coefficient smoothly shrinks towards the origin as λ increases, but also indicate that the center of each bootstrapped distribution coincides with the standard LASSO estimator based on the corresponding λ. The comparison of the three density plots in Figure 8 supports this point that the proposed penalized likelihood procedure based on the GBS can be a good alternative of standard penalized likelihood procedures in general, and the new procedure is accurate in generating bootstrap samples of a parameter and efficient in computation.

Trend Filtering via Fused LASSO. The GBS can be applied to nonparametric function inference such as a nonparametric regression model as

$$y_i = f(X_i) + \epsilon_i,$$

where $\epsilon_i \sim N(0, \sigma^2)$. A common approach to estimate the regression function f is penalized likelihood procedures that have a form of loss $\sum_{i=1}^{n}(y_i - \theta_i)^2 + \lambda\text{pen}(\boldsymbol{\theta})$, where $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_n\}^{\text{T}}$, for some penalty "pen" on $\boldsymbol{\theta}$. This class of penalized likelihood covers a wide range of nonparametric procedures, including cubic spline, smoothing spline, wavelet function estimation, fused lasso, Gaussian process regression, etc.

We consider here a trend filtering procedure to model the regression function f, and this procedure is implemented by a branch of generalized LASSO problem, so-called fused LASSO (reference). The loss function of its bootstrap procedure is expressible as

$$\hat{\boldsymbol{\theta}}_\lambda^{(b)} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^{(b)}(y_i - \theta_i)^2 + \lambda\|D\theta\|_1,$$

where D is a penalty matrix imposed on a discrete derivative operator. For example, when we penalize the k-th derivative of the regression function, the corresponding D follows

$$D = \begin{bmatrix} 1 & -1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & -1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & -1 & \ldots & 0 \\ \vdots & & & & & \end{bmatrix}, \quad \text{when } k = 0,$$

$$D = \begin{bmatrix} 1 & -2 & 1 & 0 & \ldots & 0 \\ 0 & 1 & -2 & 1 & \ldots & 0 \\ 0 & 0 & 1 & -2 & \ldots & 0 \\ \vdots & & & & & \end{bmatrix}, \quad \text{when } k = 1.$$

We note that when $k = 0$, the resulting penalized likelihood solution is a piece-wise constant function, and when $k = 1$, the solution is a piece-wise linear function. To apply the fused LASSO to the GBS, we can consider the following objective function.

$$\widehat{G} = \underset{G}{\operatorname{argmin}} \, \mathbb{E}_{\mathbf{w},\lambda} \left[\sum_{i=1}^{n} w_i(y_i - G(X_i, \mathbf{w}, \lambda))^2 + \lambda\|DG(\mathbf{X}, \mathbf{w}, \lambda)\|_1 \right], \quad (20)$$

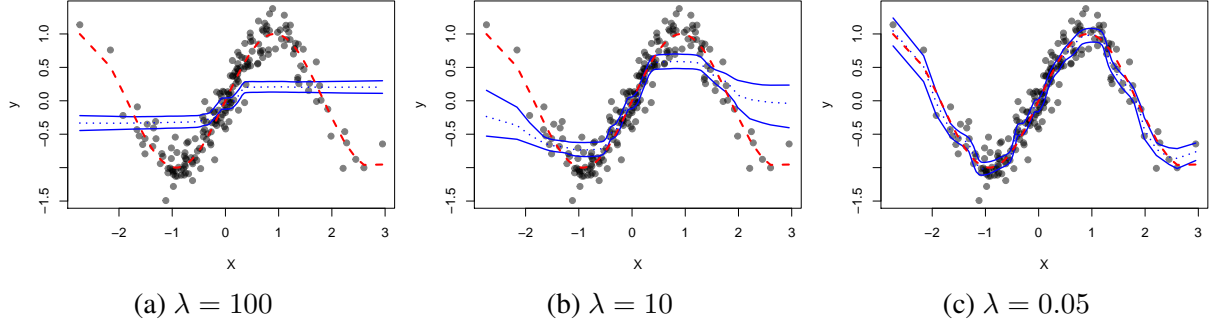(a) $\lambda = 100$ (b) $\lambda = 10$ (c) $\lambda = 0.05$

Figure 9: Evaluated by the GBS, 95% confidence bands of the trend filtering with different tuning parameters are presented. The red dashed line indicates the true regression function. The blue dotted line illustrate the mean of the bootstrap distribution.

where $G(\mathbf{X}, \mathbf{w}, \lambda) = \{G(X_1, \mathbf{w}, \lambda), \ldots, G(X_n, \mathbf{w}, \lambda)\}^{\mathrm{T}}$.

We consider a simulation setting where the true regression function $f_0(x)$ is $\sin(1.7x)$ with $\sigma^2 = 0.1$ and $k = 1$. The result of an example of this simulation setting is illustrated in Figure 9, and this figure shows how the bootstrapped confidence bands diverses as the tuning parameter λ varies. When the tuning parameter is large ($\lambda = 100$) as in Figure 9, the estimated regression function is clearly underfitted, and the shape looks like a step function. In contrast, when the tuning parameter is small ($\lambda = 0.05$), the 95% confidence band of the GBS successfully cover the true regression function. Also, this results are obtained without repetitive computations, and we just plug-in different random weights and tuning parameters into the trained generator.

4.2 Bootstrapped Cross-validation

The application of the GBS is not only restricted to tuning parameter evaluations, but it is also applicable to accelerate the computation of CV procedures. For this purpose, consider the following loss function:

$$\mathbb{E}_{\mathbf{w}'} \left[\sum_{i=1}^{n} w_i' l_\lambda(G(\mathbf{w}'_{(-I)}); y_i) \right], \tag{21}$$

where $\mathbf{w}'_{(-I)} = \{w_1', \ldots, w_n'\}$ with $w_i' = 0$ for $i \in I$ and $\{w_i' : i \notin I\}$ follows a Dirichelet distribution with uniform weights of one (or simply $w_i' = 1$ for $i \notin I$, if a bootstrap is not of interest). Because for any $i^* \in I$ the corresponding y_{i^*} does not involve the loss function due to the fact that $w_{i^*}' = 0$, the index set I and I^c can be viewed as a test data and training data indexes, respectively. Thus, the generator function G is not affected by the weights for the test set I, because the corresponding weights w_i' are exactly zero. As a result, this formalization is in accordance with out-of-sample evaluations ignoring the test data set I. This basic idea can be directly applied to construct a generator to approximate the out-of-sample prediction error. For example, when a K-fold CV is considered, we can follow some specific steps below:

Sampling weights.
1. Split the data set into K subgroups. Let us denote the indexes by $\{I_1, \ldots, I_K\}$.
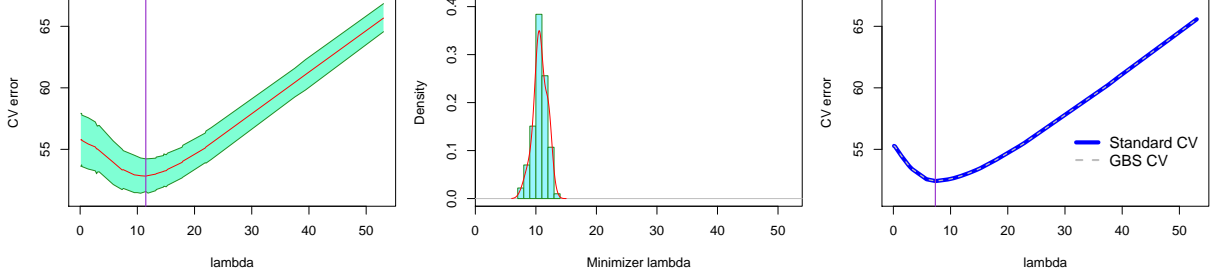2. We define a generating process of random variable $\mathbf{w}'$, involved in (21), as follows:

Figure 10: The 95% confidence band of CV error evaluated from the GBS bootstrap with random weights, and the red solid line indicates the mean curve (left); The GBS bootstrapped distribution of CV-error minimizers with respect to λ (middle); CV errors based on the standard LASSO and the GBS with a fixed weight of one (right). The purple vertical line indicates the value of λ that minimizes the CV error; $\lambda = 11.48$ (left) and $\lambda = 7.34$ (right).

     i) Randomly select one of subgroups, say I_{k^*}.

     ii) Set $w_i' = 0$, if $i \in I_{k^*}$, and the other weights $\{w_i' : i \notin I_{k^*}\}$ follow an exponential distribution with mean one. If a bootstrap is not of interest, set $w_i' = 1$ for $i \notin I_{k^*}$.

3. Choose the distribution of λ whose density values are strictly positive on Θ. A default choice is an exponential distribution with a pre-fixed scale parameter λ_0, i.e. $\lambda \sim \exp(\lambda_0)$.

Based on this set-up, a simple modification from Algorithm 1 can be used to train the generator for the K-fold CV by randomly generating λ, and then approximating the expectation in the loss function via a Monte Carlo procedure. Once the generator is trained, one can easily compute the estimated out-of-sample error across different tuning parameters. That is because the evaluation of generator is computationally efficient and fast (less than 0.1 seconds to generate 10,000 bootstrap samples in the previous examples). For some large T and a candidate set $\{\lambda_1, \ldots, \lambda_L\}$, we shall provide some steps of estimating the out-of-sample error and its uncertainty as follows:

1. For $k = 1, \ldots, K$,

     Set $t = 1$.

     i) Consider I_k and generate a weight $\mathbf{w}'$ by following the generating process in "**Sampling weights** 2. ii)".

     ii) Evaluate $\widehat{G}(\mathbf{w}', \lambda_l)$ for $l = 1, \ldots, L$, and define $\widehat{\theta}_{k,l}(\mathbf{w}') := \widehat{G}(\mathbf{w}', \lambda_l)$.

     iii) Evaluate the out-of-sample loss $r_{k,l}^{(t)} = \sum_{i \in I_k} l(\widehat{\theta}_{k,l}(\mathbf{w}'); y_i)$.

     iv) Set $t = t + 1$, and repeat i) —iii) until $t = T$.

2. Evaluate the bootstrap distribution of out-of-sample prediction errors by considering $e_l^{(t)} = \frac{1}{K} \sum_{k=1}^{K} r_{k,l}^{(t)}$.

After evaluating $e_l^{(t)}$ for $l = 1, \ldots, L$ and $t = 1, \ldots, T$, one can easily identify the bootstrapping distribution of the out-of-sample loss via the empirical distribution of $\{e_l^{(t)}\}_{t=1,\ldots,T}$ under λ_l, as well as confidence bands of the out-of-sample loss on the tuning parameter space. Moreover, let $l^{(t)} = \operatorname{argmin}_l\{e_l^{(t)}\}$, and the empirical distribution of $\lambda_{l^{(t)}}$, for $t = 1, \ldots, T$, is a bootstrap

26

distribution of the minimizer of CV errors, and it can be used to quantify uncertainty of the chosen tuning parameter via the CV (an example is given in the middle of Figure 10).

Furthermore, this bootstrap distribution of the minimizer of CV errors provides an alternative of so-called *one-standard error rule* that is commonly used with CV, in which we choose the most parsimonious model whose error is no more than one standard error above the error of the best model. In spite of its popularity, the estimated standard error is inaccurate in a sense that the value of standard error is evaluated from the sample variance of K CV errors, and K is usually chosen to be small and variability of this variance estimator is expected to be high. Instead of this unstable procedure, once we have a bootstrapping distribution of the CV solutions for the tuning parameter, we can directly estimate the standard error of CV errors as well as confidence intervals of the minimizer of CV errors. To pursue more parsimonious tuning parameter selection, we can consider an upper 95% confidence bound of the tuning parameter chosen by the CV. This procedure based on the confidence bound is more systematic way to choose a tuning parameter, but in classical frameworks of bootstrap, the computational bottleneck of the bootstrapped CV has hurdled its practical applications. In contrast, the GBS enables a practical implementation of the bootstrapped CV with a minimal extra effort in computation.

These advantages of GBS-CV are illustrated in Figure 10. The example considered in this figure is identical with one used in the previous LASSO example. The left panel shows 95% confidence bands of the CV errors across λ. As Efron and Tibshirani (1997) noted, a bootstraped CV improves the performance of prediction error estimation. However, due to heavy computational burden in the classical algorithm, the applications of the bootstrapped CV have been hindered from being utilized for modern statistical analysis with a large-sized data set. This computational issue can be overcome by the GBS, and the results show that our procedure successfully computes the confidence bands of the CV errors. The middle panel depicts the bootstrap distribution of minimizer λs of the CV errors, and for more parsimonious tuning parameter selection, we can consider an upper bound of the 95% confidence interval ($\lambda = 13.79$). We note that our GBS-based method is more systematic and principled than the one-standard error rule in a sense that the bootstrap distribution computed from the GBS directly approximates behaviors of the sampling distribution. Also, when bootstrap is not pursued and only CV is of interest, one can consider a binary random weights on $\mathbf{w}'$, not from the Dirichelet distribution, then the resulting GBS induces the same CV results with the classical procedures. This point is reflected in the right panel of Figure 10, and the CV error curve from the classical computational strategy (blue thick line) is identical to that evaluated from the GBS (grey dashed line), and they are indistinguishable.

4.3 Evaluating Permutation Null Distribution

In hypothesis testing problems of comparing relationships between some variables or different groups, evaluating the null distribution has been an essential step, and permutation techniques are commonly used to numerically approximate the distribution of the test statistics under the null. However, an implementation of the permutation test is computationally demanding, so we shall circumvent this difficulty by using the GBS idea.

We randomly generate a permutation function $d : \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$ that maps an index of observations to the other index, and we evaluate

$$\widehat{\theta}^{(b)} = \operatorname*{argmin}_{\theta} \sum_{i}^{n} l(\theta; y_{d(i)}, x_i), \tag{22}$$

27

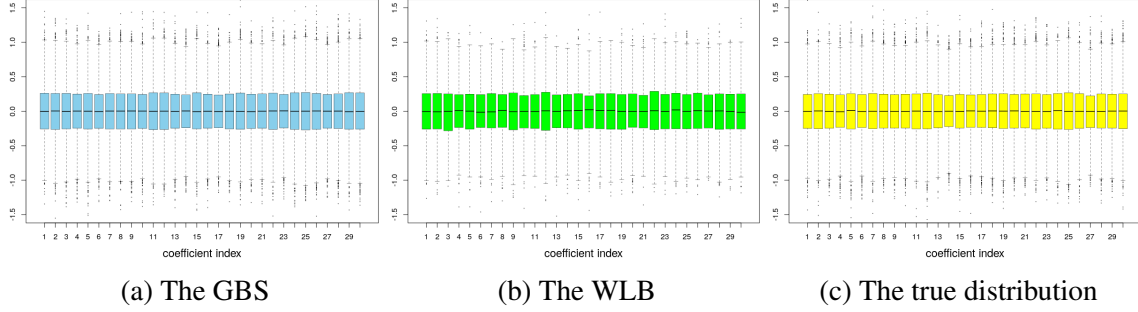| (a) The GBS | (b) The WLB | (c) The true distribution |

Figure 11: Permutation null distributions by the GBS, the WLB, and the true distribution.

and these steps are repeated for $b = 1, \dots, B$ with a large enough B. The empirical distribution of $\{\widehat{\theta}^{(1)}, \dots, \widehat{\theta}^{(B)}\}$ can be used to estimate the null distribution of a hypothesis.

The GBS formulations follows that

$$\widehat{G} = \operatorname*{argmin}_{G} \mathbb{E}_d \left[\sum_i^n l(G(\boldsymbol{d}_n); y_{d(i)}) \right], \tag{23}$$

where $d : \{1, \dots, n\} \mapsto \{1, \dots, n\}$ is a random permutation function of indexes, $\boldsymbol{d}_n$ is the vectorized binary-permutation matrix induced from d, and $\mathbb{E}_d$ indicates the expectation operator that acts on random permutations.

For linear regression models, we apply this GBS to evaluate the null distribution of a hypothesis on examining significance of the regression coefficients. That is,

$$H_0 : \boldsymbol{\theta} = 0 \text{ vs. } H_1 : \boldsymbol{\theta} \neq 0.$$

Then, the permuted null distribution can be obtained by permuting the indexes of the response $\mathbf{y}$. After evaluating the permuted null distribution, we calibrate the null distribution to be centered at the origin to follow $H_0 : \boldsymbol{\theta} = 0$. Figure 11 shows the permuted null distributions evaluated from the GBS, the classical bootstrap (WLB), and the true null distributions. The results confirm that the GBS induces an accurate bootstrap distribution without inefficient repetitions in computation.

5   Discussion

We introduced a scalable procedure for nonparametric bootstrap and random weight bootstrap by constructing a generator function of bootstrap samples. We showed the proposed procedure GBS accelerates bootstrap procedures, as well as other repetitive computational procedures such as CV procedures, at a fold of hundreds compared to classical bootstrap procedures.

Even though we applied the GBS to nonparametric bootstrap settings, this idea can be extended to parametric bootstrap procedures, However, we note that applicable setting for parameteric bootstrap is limited to a case where a explicit functional form of the data-generating process is available. An example of this case is linear models. When a regression coefficient $\boldsymbol{\theta}^*$ is given, the response available can be generated by $y_i^* = X_i \boldsymbol{\theta} + \sigma z_i$, where $z_i \sim N(0, I_p)$. Then, the corresponding GBS can be obtained as follows:

$$\widehat{G} = \operatorname*{argmin}_{G} \mathbb{E}_{\boldsymbol{z}} \left[\sum_{i=1}^n \left(X_i \hat{\boldsymbol{\theta}}_{MLE} + \sigma z_i - X_i G(\boldsymbol{z}) \right)^2 \right].$$

28

where $\boldsymbol{\theta}_{MLE}$ is the MLE of the regression coefficient. For more generalized application of the GBS to parametric bootstrap, further research and development are demanded in future.

References

Aghazadeh, A., Spring, R., Lejeune, D., Dasarathy, G., Shrivastava, A., and Baraniuk, R. (2018). MISSION: Ultra large-scale feature selection using count-sketches. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 80–88, Stockholmsmässan, Stockholm Sweden. PMLR.

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

Barbe, P. and Bertail, P. (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.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Bray, N. L., Pimentel, H., Melsted, P., and Pachter, L. (2016). Near-optimal probabilistic rna-seq quantification. *Nature biotechnology*, 34(5):525–527.

Carbonetto, P. and Stephens, M. (2012). Scalable variational inference for Bayesian variable selection in regression, and its accuracy in genetic association studies. *Bayesian Analysis*, 7(1):73–108.

Carlstein, E., Do, K.-A., Hall, P., Hesterberg, T., Künsch, H. R., et al. (1998). Matched-block bootstrap for dependent data. *Bernoulli*, 4(3):305–328.

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

Choi, Y., Choi, M., Kim, M., Ha, J.-W., Kim, S., and Choo, J. (2018). Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797.

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

Davison, A. C. and Hinkley, D. V. (1997). *Bootstrap methods and their application*, volume 1. Cambridge university press.

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

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

Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Fan, J. and Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *J. Am. Statist. Ass.*, 96(456):1348–1360.

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

Giné, E. and Zinn, J. (1990). Bootstrapping general empirical measures. *The Annals of Probability*, pages 851–869.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 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*, pages 1431–1452.

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

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. (2019). Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pages 2722–2730.

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

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134.

Jeffreys, H. (1961). Theory of probability.

Johnson, V. E. and Rossell, D. (2010). On the use of non-local prior densities in Bayesian hypothesis tests. *J. R. Statist. Soc. B*, 72(2):143–170.

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

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *J. Am. Statist. Ass.*, 90(430):773–795.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751.

Kirk, P. D. and Stumpf, M. P. (2009). Gaussian process regression bootstrapping: exploring the effects of uncertainty in time course data. *Bioinformatics*, 25(10):1300–1306.

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

Kocherginsky, M., He, X., and Mu, Y. (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*, pages 349–359. Springer.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., 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*, pages 4681–4690.

Liang, F., Paulo, R., Molina, G., Clyde, M. A., and Berger, J. O. (2008). Mixtures of g priors for Bayesian variable selection. *J. Am. Statist. Ass.*, 103(481):410–423.

MacKay, D. J. (2003). Information theory, inference and learning algorithms. chapter 33. Cambridge university press.

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

Pati, D., Bhattacharya, A., and Yang, Y. (2018). On statistical optimality of variational bayes. In *International Conference on Artificial Intelligence and Statistics*, pages 1579–1588.

Politis, D. N., Romano, J. P., and Wolf, M. (2001). On the asymptotic theory of subsampling. *Statistica Sinica*, pages 1105–1124.

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

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian Process for Machine Learning*. MIT Press, Cambridge.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538.

Rousseeuw, P. J. and Leroy, A. M. (2005). *Robust regression and outlier detection*, volume 589. John wiley & sons.

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

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

Shao, J. and Tu, D. (2012). *The jackknife and bootstrap*. Springer Science & Business Media.

Silverman, B. and Young, G. (1987). The bootstrap: to smooth or not to smooth? *Biometrika*, 74(3):469–479.

Street, J. O., Carroll, R. J., and Ruppert, D. (1988). A note on computing robust regression estimates via iteratively reweighted least squares. *The American Statistician*, 42(2):152–154.

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

Vervier, K., Mahé, P., Tournoud, M., Veyrieras, J.-B., and Vert, J.-P. (2016). Large-scale machine learning for metagenomics sequence classification. *Bioinformatics*, 32(7):1023–1032.

Wahba, G. (1978). Improper priors, spline smoothing and the problem of guarding against model errors in regression. *J. R. Statist. Soc. B*, pages 364–372.

Wahba, G. (1990). *Spline models for observational data*. Society for Industrial and Applied Mathematics.

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

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. (2009). Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120.

Wood, D. E. and Salzberg, S. L. (2014). Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15(3):R46.

Yao, Y., Vehtari, A., Simpson, D., and Gelman, A. (2018). Yes, but did it work?: Evaluating variational inference. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5581–5590, Stockholmsmässan, Stockholm Sweden. PMLR.

Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.

Zhang, C.-H. (2010). Nearly unbiased variable selection under minimax concave penalty. *Ann. Statist.*, 38(2):894–942.