Scalable Semi-Supervised SVM via Triply Stochastic Gradients

Xiang Geng[1] , **Bin Gu**[1,2] , **Xiang Li**[4] , **Wanli Shi**[1] , **Guansheng Zheng**[1] and **Heng Huang**[2,3] *
[1]School of Computer & Software, Nanjing University of Information Science & Technology, P.R.China
[2]JD Finance America Corporation
[3]Department of Electrical & Computer Engineering, University of Pittsburgh, USA
[4]Computer Science Department, University of Western Ontario, Canada
gengxiang@nuist.edu.cn, jsgubin@gmail.com, lxiang2@uwo.ca, wanlishi@nuist.edu.cn,
zgs@nuist.edu.cn, heng.huang@pitt.edu

Abstract

Semi-supervised learning (SSL) plays an increasingly important role in the big data era because a large number of unlabeled samples can be used effectively to improve the performance of the classifier. Semi-supervised support vector machine (S^{3}VM) is one of the most appealing methods for SSL, but scaling up S^{3}VM for kernel learning is still an open problem. Recently, a doubly stochastic gradient (DSG) algorithm has been proposed to achieve efficient and scalable training for kernel methods. However, the algorithm and theoretical analysis of DSG are developed based on the convexity assumption which makes them incompetent for non-convex problems such as S^{3}VM. To address this problem, in this paper, we propose a triply stochastic gradient algorithm for S^{3}VM, called TSGS^{3}VM. Specifically, to handle two types of data instances involved in S^{3}VM, TSGS^{3}VM samples a labeled instance and an unlabeled instance as well with the random features in each iteration to compute a triply stochastic gradient. We use the approximated gradient to update the solution. More importantly, we establish new theoretic analysis for TSGS^{3}VM which guarantees that TSGS^{3}VM can converge to a stationary point. Extensive experimental results on a variety of datasets demonstrate that TSGS^{3}VM is much more efficient and scalable than existing S^{3}VM algorithms.

1 Introduction

Semi-supervised learning (SSL) plays an increasingly important role in the big data era because a large number of unlabeled samples can be used effectively to improve the performance of the classifier. Semi-supervised support vector machine (S^{3}VM) [Bennett and Demiriz, 1999] is one of the most appealing methods for SSL. Specifically, S^{3}VM enforces the classification boundary to go across the less-dense regions in the reproducing kernel Hilbert space (RKHS), while keeping the labeled data correctly classified. Unfortunately, this will lead to a non-convex optimization problem. It is well known that solving a non-convex optimization problem is normally difficult than solving a convex one like standard support vector machine. Under this arduous challenge, a wide spectrum of methods for S^{3}VM have been proposed in the last two decades. Generally speaking, these methods can be roughly divided into three groups, *i.e.*, methods with self-labeling heuristics, concave-convex procedure (CCCP) methods and gradient-based methods. We give a brief review of these representative S^{3}VM methods in Section 2 and Table 1.

Unfortunately, these traditional S^{3}VM methods are inefficient due to increased complexity introduced by the cost of kernel computation as well as non-convexity. Specifically, the kernel matrix needs $O(n^2d)$ operations to be calculated and $O(n^2)$ memory to be stored, where n denotes the size of training samples and d denotes dimension of the data [Gu *et al.*, 2018b]. Essentially, gradient-based S^{3}VM methods have $O(n^3)$ complexity due mainly to the operations on the kernel matrix. Even though a convex kernel problem can be solved by a state-of-the-art solver (*e.g.* LIBSVM), $O(n^\kappa)$ computation is still needed where $1 < \kappa < 2.3$ [Chang and Lin, 2011]. While to handle the non-convexity of S^{3}VM, the methods using self-labeling heuristics and CCCP-based algorithms need to solve multiple convex sub-problems to guarantee that they finally converge [Yuille and Rangarajan, 2002]. As a result, these methods scale as $O(tn^\kappa)$, where t denotes the number of solving sub-problems. We summarize the computational complexities and memory requirements of the representative S^{3}VM methods in Table 1. As pointed in [Gu *et al.*, 2018d], scaling up S^{3}VM is still an open problem.

Recently, a novel doubly stochastic gradient (DSG) method [Dai *et al.*, 2014] was proposed to achieve efficient and scalable training for kernel methods. Specifically, in each iteration, DSG computes a doubly stochastic gradient by sampling a random data sample and the corresponding random features to update the solution. Thus, DSG avoids computing and storing a kernel matrix, while enjoying nice computational and space complexities. Xie et al. [2015] used DSG to scale up nonlinear component analysis. To the best of our knowledge, [Xie *et al.*, 2015] is the only work based on DSG to solve a non-convex problem.

However, existing algorithms and theoretical analysis of DSG cannot be applied to S^{3}VM due to the following two reasons. **1) Multiple data distributions:** S^{3}VM minimizes the training errors coming from two different sources. One

*To whom all correspondence should be addressed.

Algorithm	Reference	Method	Computational Complexity	Space Complexity
S^3VM^{light}	[Joachims, 1999]	Self-labeling heuristics	$O(tn^\kappa)$	$O(n^2)$
NTS^3VM	[Chapelle, 2007]	Gradient-based	$O(n^3)$	$O(n^2)$
BGS^3VM	[Le *et al.*, 2016]	Gradient-based	$O(n^3)$	$O(n^2)$
BLS^3VM	[Collobert *et al.*, 2006]	CCCP-based	$O(tn^\kappa)$	$O(n^2)$
ILS^3VM	[Gu *et al.*, 2018d]	CCCP-based	$\approx O(Tn^2)$	$O(n^2)$
$TSGS^3VM$	Our	TSG	$O(mT^2)$	$O(T)$

Table 1: Comparisons of computational complexities and memory requirements of representative S^3VM algorithms. (n is the training size, T is the total number of iteration, t denotes the number of outer loops and $1 < \kappa < 2.3$)

is the expected error on the unlabeled data, and the other one is the mean error on the labeled data whose size is normally significantly smaller than the one of unlabeled data. However, DSG only considers the expected error on the labeled data.
2) Non-convexity analysis: The theoretical analysis in [Xie *et al.*, 2015] is based on a strong assumption (*i.e.*, the initialization needs to be close to the optimum). However, such an assumption is rarely satisfied in practice. Besides, they focus on the nonlinear component analysis instead of general non-convex problems. Thus, it is infeasible to extend the analysis of [Xie *et al.*, 2015] to S^3VM.

To address this challenging problem, we first propose a new and practical formulation of S^3VM. Then, we develop a new triply stochastic gradient algorithm ($TSGS^3VM$) to solve the corresponding optimization problem. Specifically, to handle two types of data instances involved in S^3VM, $TSGS^3VM$ samples a labeled instance and an unlabeled instance as well with their random features in each iteration to compute a triply stochastic gradient (TSG). We then use the TSGs to iteratively update the solution. A critical question is whether and how fast this optimization process with multiple randomness would converge. In addressing this concern, we establish new theoretic analysis for $TSGS^3VM$ which guarantees that $TSGS^3VM$ can converge to a stationary point with a sublinear convergence rate for a general non-convex learning problem under weak assumptions. Extensive experimental results demonstrate the superiority of $TSGS^3VM$.
Novelties. We summary the main novelties of this paper as follows.

- To scale up S^3VM, we propose a practical formulation of S^3VM and develop a novel extension of DSG that could solve optimization problems with multiple data sources.
- We have established the new theoretic analysis of $TSGS^3VM$ algorithm for a general non-convex learning problem which guarantees its convergence to a stationary point. To the best of our knowledge, it is the first work offering non-convex analysis for DSG-like algorithms without initialization assumption.

2 Related Works

We give a brief review of kernel approximation methods as well as the representative S^3VM methods.

Kernel Approximation. There are many kernel approximation methods proposed to address the scalability issue of kernel methods. For instance, low-rank factors are used to approximate the kernel matrix in [Drineas and Mahoney, 2005]. Rahimi & Recht [2008] provided another method that uses random features to approximate the map function explicitly. However, as analyzed in [Drineas and Mahoney, 2005; Lopez-Paz *et al.*, 2014], the rank for low-rank and the number of random features need to be $O(n)$ to obtain a good generalization ability. To further improve the random features method, Dai et al. [2014] proposed DSG descent algorithm. Carratino et al. [2018] proved that DSG only need $O(\sqrt{n})$ random features to obtain a good result. However, existing DSG methods [Li *et al.*, 2017; Gu *et al.*, 2018c] can not be used for S^3VM as discussed previously.

S^3VM Methods. As mentioned above, traditional S^3VM methods can be roughly divided into three types, *i.e.*, the method of self-labeling heuristics, the concave-convex procedure (CCCP) method, and the gradient-based method. For the method of self-labeling heuristics, Joachims [1999] proposed a S^3VM^{light} algorithm which uses self-labeling heuristics for labeling the unlabeled data, then iteratively solve this standard SVM until convergence. CCCP-based methods were proposed to solve S^3VM in [Chapelle and Zien, 2005; Wang *et al.*, 2007]. The basic principle of CCCP is to linearize the concave part of S^3VM's objective function around a solution obtained in the current iteration so that sub-problem is convex. Then the CCCP framework solves a sequence of the convex sub-problem iteratively until decision variable converges. Based on CCCP framework, Gu et al. [Gu *et al.*, 2018d] proposed an incremental learning method for S^3VM which is suitable for the online scenario. For gradient-based methods, Chapelle and Zien [2005] approximate the kernel matrix K using low-rank factors, then using gradient descent to solve S^3VM on the low-rank matrix. BGS^3VM [Le *et al.*, 2016] uses budgeted SGD to limit the model size to two predefined budgets B_l and B_u.

3 Preliminaries

In this section, we first give a general non-convex learning problem for S^3VM, and then give a brief review of random feature approximation.

3.1 S^3VM Optimization Formulation

Given the training dataset $\mathcal{X}$ constituted with n^l labeled examples $\mathcal{L} := \{(x_i, y_i)\}_{i=1}^{n^l}$ and n^u unlabeled examples $\mathcal{U} :=$

Name	$u(r)$	$u'(r)$
SHG	$\max\{0, 1-\|r\|\}$	$\begin{cases} 0 & \text{if } \|r\| \geq 1 \\ -1 & \text{if } \|r\| < 1 \end{cases}$
SSHG	$\frac{1}{2}\max\{0, 1-\|r\|\}^2$	$\begin{cases} 0 & \text{if } \|r\| \geq 1 \\ \|r\| - 1 & \text{if } \|r\| < 1 \end{cases}$
Ramp	$H_1(r) - H_s(r)$	$H_1'(r) - H_s'(r)$
DA	$\exp(-5r^2)$	$-10r \cdot \exp(-5r^2)$

Table 2: Summary of the non-convex loss functions used in S^{3}VM, where $H_s(\cdot) = \max\{0, s - \cdot\}$, then $H_s'(\cdot) = 0$, if $\cdot \geq s$ else $H_s'(\cdot) = -1$. SHG, SSHG and DA denote symmetric hinge, square SHG and a differentiable approximation to SSHG respectively.

$\{x_i\}_{i=n^l+1}^{n}$, where $n = n^l + n^u$, $x_i \in \mathbb{R}^d$, and $y_i \in \{1, -1\}$. Traditional S^{3}VM solves the following problem.

$$\min_{f \in \mathcal{H}} \quad \frac{1}{2}||f||_{\mathcal{H}}^2 + \frac{C}{n^l} \sum_{(x,y)\in\mathcal{L}} l(f(x), y) + \frac{C^*}{n^u} \sum_{x \in \mathcal{U}} u(f(x))$$

where C and C^* are regularization parameters, $||\cdot||_{\mathcal{H}}$ denotes the norm in RKHS, $l(r, v) = \max(0, 1 - vr)$ is the hinge loss, its subgradient $l'(r, v) = 0$, if $rv \geq 1$, else $l'(r, v) = -v$, $u(r)$ is the non-convex loss function which enforce unlabeled data away from the discrimination hyperplane. We summarize the commonly used non-convex S^{3}VM losses and its subgradient $u'(r)$ in Table 2.

For S^{3}VM problems, however, the volumes of labeled and unlabeled data are usually quite different. Because of the labeling cost, the labeled dataset is often very small, while a large amount of unlabeled data can be obtained relatively easily. Taking this into consideration, we propose to solve a novel S^{3}VM formulation as follows.

$$\begin{aligned} & \min_{f \in \mathcal{H}} R(f) \qquad (1) \\ = \quad & \frac{1}{2}||f||_{\mathcal{H}}^2 + \frac{C}{n^l} \sum_{(x,y)\in\mathcal{L}} l(f(x), y) + C^* \mathbb{E}_{x\sim P(x)} u(f(x)) \end{aligned}$$

where $P(x)$ denotes the target data distribution. Notice that we use the empirical mean error on the labeled dataset, while using the expected error on the whole distribution for the unlabeled data.

3.2 Random Feature Approximation

Random feature is a powerful technique to make kernel methods scalable. It uses the intriguing duality between kernels and stochastic processes. Specifically, according to the Bochner theorem [Wendland, 2004], for any positive definite PD kernel $k(\cdot, \cdot)$, there exists a set Ω, a probability measure $\mathbb{P}$ and a random feature map $\phi_\omega(x)$, such that $k(x, x') = \int_\Omega \phi_\omega(x)\phi_\omega(x')d\mathbb{P}(\omega)$. In this way, the value of the kernel function can be approximated by explicitly computing random features $\phi_\omega(x) = [\frac{1}{\sqrt{m}}\phi_{\omega_1}(x), \frac{1}{\sqrt{m}}\phi_{\omega_2}(x), \cdots, \frac{1}{\sqrt{m}}\phi_{\omega_m}(x)]$, *i.e.*,

$$k(x, x') \approx \frac{1}{m} \sum_{i=1}^{m} \phi_{\omega_i}(x)\phi_{\omega_i}(x') \qquad (2)$$

where m is the number of random features. Using Gaussian RBF kernel as a concrete example, it yields a Gaussian distribution $\mathbb{P}(\omega)$ over random feature maps of Fourier basis functions $\phi_{\omega_i}(x) = \sqrt{2}cos(\omega_i^T x + b)$ to compute its feature mapping, where ω_i is drawn from $\mathbb{P}(\omega)$ and b is drawn uniformly form $[0, 2\pi]$. Moreover, many random feature construction methods have been proposed for various kernels, such as dot-product kernels and Laplacian kernels.

The theory of RKHS provides a rigorous mathematical framework for studying optimization problems in the functional space. Specifically, we know that every PD kernel $k(x, x')$ has a corresponding RKHS $\mathcal{H}$. An RKHS $\mathcal{H}$ has the reproducing property, *i.e.*, $\forall x \in \mathcal{X}, \forall f \in \mathcal{H}$, we always have $\langle f(\cdot), k(x, \cdot)\rangle_{\mathcal{H}} = f(x)$. Besides, functional gradient in RKHS $\mathcal{H}$ can be computed as $\nabla f(x) = k(x, \cdot)$ and $\nabla||f||_{\mathcal{H}}^2 = 2f$.

4 Triply Stochastic S^{3}VM

The above section has introduced the basic theoretic tools for triply stochastic functional gradient descent. Now we introduce how to utilize these tools to solve the S^{3}VM problem.

4.1 Triply Stochastic Gradient

From Eq. (1), it is not hard to notice that $R(f)$ involves two different data sources. Taking into consideration the distribution of random features $\omega \sim \mathbb{P}(\omega)$ would give us three sources of possible randomness. Here we will show how to explicitly compute the stochastic gradient with these three sources of randomness.

Stochastic Functional Gradients. Naturally, to iteratively update f in a stochastic manner, we need to sample instances from the labeled dataset as well as the whole distribution. Different from DSG, we here randomly sample a pair of data points, from the labeled and the unlabeled data distributions, respectively. Then we can obtain stochastic functional gradients for $R(f)$ with these two data points as follow,

$$g(\cdot) = f(\cdot) + \xi(\cdot) \qquad (3)$$

where $\xi(\cdot)$ is the gradient contributed by the loss from both labeled and unlabeled data. It can be computed using the chain rule

$$\xi(\cdot) = Cl'(f(x^l), y^l)k(x^l, \cdot) + C^* u'(f(x^u))k(x^u, \cdot) \qquad (4)$$

where x^l, x^u are sampled from the labeled dataset and unlabeled distribution $P(x)$ respectively. Next we will plugging the random feature approximation technique described in the previous section.

Random Feature Approximation. According to Eq. (2), when we use stochastically generated random feature ω, we can further approximate $\xi(\cdot)$ as:

$$\begin{aligned} & \xi(\cdot) \approx \zeta(\cdot) \qquad (5) \\ = & Cl'(f(x^l), y^l)\phi_\omega(x^l)\phi_\omega(\cdot) + C^* u'(f(x^u))\phi_\omega(x^u)\phi_\omega(\cdot) \end{aligned}$$

note that $\xi(\cdot) = \mathbb{E}_\omega[\zeta(\cdot)]$. This leads to an unbiased estimator of the original functional gradient with three layers of stochasticity, *i.e.*,

$$\nabla R(f) = \mathbb{E}_{(x^l,y^l)\in\mathcal{L}} \mathbb{E}_{x^u \sim P(x)} \mathbb{E}_\omega(\zeta(\cdot)) + f(\cdot) \qquad (6)$$

Since three random events occur per iteration, *i.e.* x^l, x^u, ω, we call our approximate functional gradient as triply stochastic functional gradient.

Update Rules. In the t-th iteration, the triply stochastic (functional) gradient update rule for f is:

$$f_{t+1}(\cdot) = f_t(\cdot) - \gamma_t(\zeta_t(\cdot) + f_t(\cdot)) = \sum_{i=1}^{t} a_t^i \zeta_i(\cdot) \quad (7)$$

where γ denotes the step size and the initial value $f_1(\cdot) = 0$. It is straight forward to calculate that $a_t^i = -\gamma_i \prod_{k=i+1}^{t}(1-\gamma_k)$. Ideally, if we could somehow compute the stochastic (functional) gradients $\xi_j(\cdot)$, the update rule becomes:

$$h_{t+1}(\cdot) = h_t(\cdot) - \gamma_t(\xi_t(\cdot) + h_t(\cdot)) = \sum_{i=1}^{t} a_t^i \xi_i(\cdot) \quad (8)$$

where we have used h_{t+1} instead of f_{t+1} to distinguish from the triply stochastic (functional) gradient update rule and $h_1(\cdot) = 0$. However, to avoid the expense of kernel computation, our algorithm will use the triply stochastic update rule Eq. (7) instead of Eq. (8).

4.2 Algorithm

Based on the above triply stochastic gradient update rules (7), we provide the TSGS^{3}VM training and prediction procedures in Algorithms 1 and 2 receptively. Notice that directly computing all the random features still needs a large amount of memory. Following the pseudo-random number generators setting of [Dai *et al.*, 2014], our random feature generator is initialized by a predefined seed according to iteration. Thus, TSGS^{3}VM does not need to save the random feature matrix which makes it more memory friendly. In the i-th iteration, our method will execute the following steps.

1. Sample data pair (lines 2-3 in Algorithm 1): Stochastically sample a labeled sample (x_i^l, y_i^l) and an unlabeled sample x_i^u from different distribution respectively.
2. Sample random features (line 4 in Algorithm 1): Stochastically sample $\omega^i \sim \mathbb{P}(\omega)$ with seed i and generate random features.
3. Update coefficients (lines 5-8 in Algorithm 1): Evaluate function value and update f according to Eq. (7).

Remark 1 *For each iteration, TSGS^{3}VM needs $O(mT)$ operations to evaluate function value, since evaluating the function value needs generating m random features ($O(m)$) for T times. Thus, the total computational complexity of TSGS^{3}VM is $O(mT^2)$. Due to the use of random features and pseudo-random method, TSGS^{3}VM only requires $O(T)$ memory, where T is the iteration number.*

5 Theoretical Guarantees

We follow the common goal of non-convex analysis [Ghadimi and Lan, 2013; Gu *et al.*, 2018a; Huo *et al.*, 2018] to bound $\mathbb{E}||\nabla R(f)||^2$, which means that the objective function will converge (in expectation) to a stationary point f_*. When we use the hypothetical update rule (8), h_t will always be inside

Algorithm 1 TSGS^{3}VM Train

Input: $\mathcal{L}, P(x), \mathbb{P}(\omega), \phi_\omega(x), u(f(x)), C, C^*$
1: **for** $i = 1, \cdots, T$ **do**
2: Sample $(x_i^l, y_i^l) \sim \mathcal{L}$
3: Sample $x_i^u \sim P(x)$
4: Sample $\omega_i \sim \mathbb{P}(\omega)$ with seed i
5: $f(x_i^l) =$ **Predict**$(x_i^l, \{\alpha_j\}_{j=1}^{i-1})$
6: $f(x_i^u) =$ **Predict**$(x_i^u, \{\alpha_j\}_{j=1}^{i-1})$
7: $\alpha_i = -\gamma_i(Cl'(f(x_i^l), y_i^l)\phi_{\omega_i}(x_i^l) + C^* u'(f(x_i^u))\phi_{\omega_i}(x_i^u))$
8: $\alpha_j = (1-\gamma_i)\alpha_j$, for $j = 1, \cdots, i-1$
9: **end for**
Output: $\{\alpha_i\}_{i=1}^T$.

Algorithm 2 TSGS^{3}VM Predict

Input: $\mathbb{P}(\omega), \phi_\omega(x), x, \{\alpha_i\}_{i=1}^T$
1: Set $f(x) = 0$
2: **for** $i = 1, \cdots, T$ **do**
3: Sample $\omega_i \sim \mathbb{P}(\omega)$ with seed i
4: $f(x) = f(x) + \alpha_i \phi_{\omega_i}(x)$
5: **end for**
Output: $f(x)$

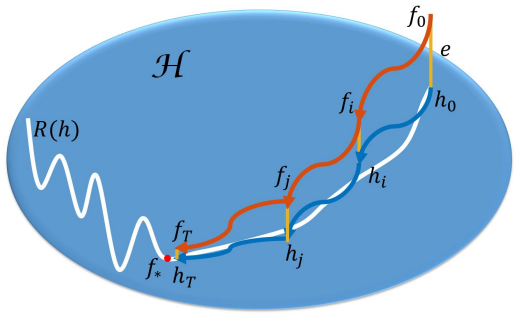
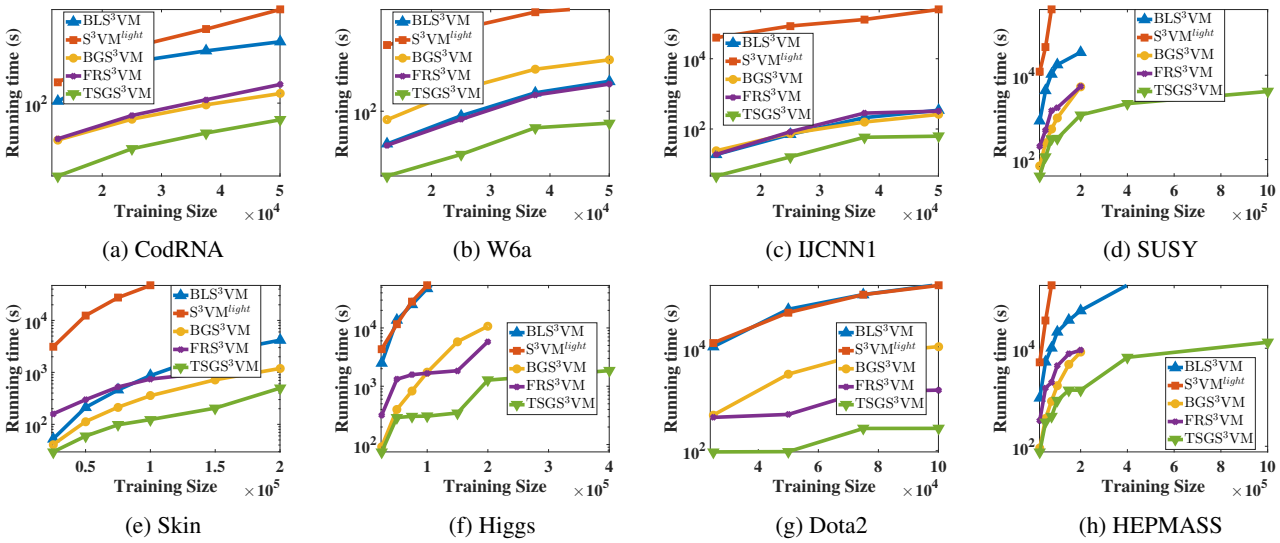


Figure 1: Illustration of how TSGS^{3}VM converge to a stationary point, where e denotes for the error between f_t and h_t, the white line denote the objective value $R(h)$. In this toy model we assume all horizontal points in $\mathcal{H}$ have the same objective value.

of $\mathcal{H}$. However, because we could only use random features to approximate h_t with f_t, we face the risk that functional f_t could be outside of $\mathcal{H}$. As a consequence, $\mathbb{E}||\nabla R(f)||_{\mathcal{H}}^2 = 0$ is not the stationary point of the objective function (1). From Eq. (7) and Eq. (8), it is obvious that every update of h_t happens implicitly with an update of $f_t(x)$. According to this relationship, we proposed to divide the analysis in two parts. As illustrated in Fig. 1, for a general non-convex optimization problem $R(h)$, we prove that the h_{t+1} converges to a stationary point f_* (*i.e.*, $\mathbb{E}||\nabla R(h_{t+1})||_{\mathcal{H}}^2 < \epsilon_1$) firstly. Then we prove that $f_{t+1}(x)$ keeps close to its hypothetic twin $h_{t+1}(x)$ for any $x \in \mathcal{X}$ (*i.e.*, $|f_{t+1}(x) - h_{t+1}(x)|^2 < \epsilon_2$).

Our analysis is built upon the following assumptions which are standard for the analysis of non-convex optimization and DSG [Dai *et al.*, 2014].

Assumption 1 *(Lipschitzian gradient) The gradient function*



(a) CodRNA (b) W6a (c) IJCNN1 (d) SUSY

(e) Skin (f) Higgs (g) Dota2 (h) HEPMASS

Figure 2: Running time of different S^{3}VM solvers v.s. training size on the eight benchmark data sets, where the lines of BLS^{3}VM and S^{3}VM^{light} are incomplete on several datasets due to the corresponding implementations crash on the large-scale training set.

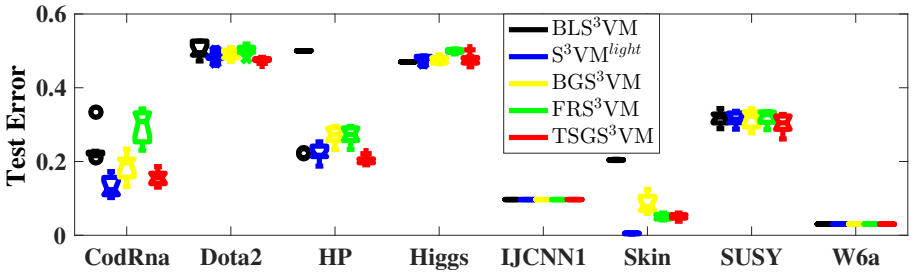


Figure 3: The boxplot of test error for different methods on different datasets.

Dataset	Dimensionality	Samples	Source
CodRNA	8	59,535	LIBSVM
W6a	300	49749	
IJCNN1	22	49,990	
SUSY	18	5,000,000	
Skin	3	245,057	
Higgs	28	1,100,000	
Dota2	16	102,944	UCI
HEPMASS	28	10,500,000	

Table 3: Datasets used in the experiments.

$\nabla R(f)$ *is Lipschitzian, that is to say*

$$||\nabla R(f) - \nabla R(g)||_{\mathcal{H}} \leq L||f - g||_{\mathcal{H}}, \forall f, g \in \mathcal{H} \quad (9)$$

Assumption 2 *(Lipschitz continuity)* $l(r, v)$ *is* L'***-Lipschitz continuous*** *in terms of its 1st argument.* $u(r, v)$ *is* U'***-Lipschitz continuous*** *in terms of its 1st argument. We further denote* $M' = CL' + C^*U'$.

Assumption 3 *(Bound of derivative) The derivatives are bounded:* $|l'| < M^l$ *and* $|u'| < M^u$*, where* l' *and* u' *is the derivative of* $l(r, v)$ *and* $u(r)$ *w.r.t. the 1st argument respectively. We further denote* $M = CM^l + C^*M^u$.

Assumption 4 *(Bound of kernel and random features) We have an upper bound for the kernel value,* $k(x, x') \leq \kappa$*. There is an upper bound of random feature norm,* i.e.*,* $|\phi_\omega(x)\phi_\omega(x')| \leq \phi$.

Suppose the total number of iterations is T, we introduce our main theorems as below. All the detailed proofs are provided in our Appendix[1].

Theorem 1 *For any* $x \in \mathcal{X}$*, fix* $\gamma = \frac{\theta}{T^{3/4}}$ *with* $0 < \theta \leq T^{3/4}$*, we have*

$$\mathbb{E}_{x^l, x^u, \omega_t}\left[\left|f_t(x) - h_t(x)\right|^2\right] \leq \frac{D}{T^{1/2}} \quad (10)$$

where $D = \theta^2 M^2(\sqrt{\kappa} + \sqrt{\phi})^2$.

Remark 2 *The error between* f_{t+1} *and* h_{t+1} *is mainly induced by random features. Theorem 1 shows that this error has the convergence rate of* $O(1/\sqrt{T})$ *with proper step size.*

Theorem 2 *For fixed* $\gamma = \frac{\theta}{T^{3/4}}$*,* $0 < \theta \leq T^{3/4}$*, we have that*

$$\mathbb{E}_{x^l, x^u, \omega_t}[||\nabla R(h_t)||^2_{\mathcal{H}}] \leq \frac{E}{T^{1/4}} + \frac{F}{T^{3/4}} \quad (11)$$

[1]The Appendix is available at https://drive.google.com/open?id=1iVE5hMJ-DbA87K9skuvL4c250OMBD4EG.

where $E = \frac{1}{\theta}[R(h_1) - R^*] + \theta M^2 M'(\sqrt{\kappa} + \sqrt{\phi})\kappa, F = 2\theta M^2 L\kappa$, R^* *denotes the optimal value of (1).*

Remark 3 *Instead of using the convexity assumption in [Dai* et al.*, 2014], Theorem 2 uses Lipschitzian gradient assumption to build the relationship between gradients and the updating functions* h_{t+1}*. Thus, we can bound each error term of* $\mathbb{E}_{x^l,x^u,\omega_t}[||\nabla R(h_t)||^2_{\mathcal{H}}]$ *as shown in Appendix. Note that compared to the strong assumption (*i.e.*, the good initialization) used in [Xie* et al.*, 2015], the assumptions used in our proofs are weaker and more realistic.*

6 Experiments and Analysis

In this section, we will evaluate the practical performance of TSGS^{3}VM when comparing against other state-of-the-art solvers.

6.1 Experimental Setup

To show the advantage our TSGS^{3}VM for large-scale S^{3}VM learning, we conduct the experiments on large scale datasets to compare TSGS^{3}VM with other state-of-the-art algorithms in terms of predictive accuracy and time consumption. Specifically, the compared algorithms in our experiments are summarized as follows[2].

1. ***BLS^{3}VM*** [Collobert *et al.*, 2006]: The state-of-art S^{3}VM algorithm based on CCCP and SMO algorithm [Cai and Cherkassky, 2012].
2. ***S^{3}VM^{light}*** [Joachims, 1999]: The implementation in the popular S^{3}VM^{light} software. It is based on the local combinatorial search guided by a label switching procedure.
3. ***BGS^{3}VM***[Le *et al.*, 2016]: Our implementation of BGS^{3}VM in MATLAB.
4. ***FRS^{3}VM***: Standard SGD with fixed random features.
5. ***TSGS^{3}VM***: Our proposed S^{3}VM algorithm via triply stochastic gradients.

Implementation. We implemented the TSGS^{3}VM algorithm in MATLAB. For the sake of efficiency, our TSGS^{3}VM implementation also uses a mini-batch setting. We perform experiments on Intel Xeon E5-2696 machine with 48GB RAM. The Gaussian RBF kernel $k(x, x') = \exp(-\sigma||x - x'||^2)$ and the loss function $u = \max\{0, 1 - |r|\}$ was used for all algorithms. 5-fold cross-validation was used to determine the optimal settings (test error) of the model parameters (the regularization factor C and the Gaussian kernel parameter σ), the parameters C^* was set to $C\frac{n^l}{n^u}$. Specifically, the unlabeled dataset was divided evenly to 5 subsets, where one of the subsets and all the labeled data are used for training, while the other 4 subsets are used for testing. Parameter search was done on a 7×7 coarse grid linearly spaced in the region $\{\log_{10} C, \log_{10} \sigma)| - 3 \leq \log_{10} C \leq 3, -3 \leq \log_{10} \sigma \leq 3\}$ for all methods. For TSGS^{3}VM, the step size γ equals $\frac{1}{\eta}$, where $0 \leq \log_{10} \eta \leq 3$ is searched after C and σ. Besides, the number of random features is set to be $\lceil\sqrt{n}\rceil$ and the batch size is set to 256. The test error was obtained by using these optimal model parameters for all the methods. To achieve a comparable accuracy to our TSGS^{3}VM, we set the minimum budget sizes B_l and B_u as 100 and $0.2 * n_u$ respectively for BGS^3VM. We stop TSGS^{3}VM and BGS^3VM after one pass over the entire dataset. We stop FRS^{3}VM after 10 pass over the entire dataset to achieve a comparable accuracy. All results are the average of 10 trials.

Datasets. Table 3 summarizes the 8 datasets used in our experiments. They are from LIBSVM[3] and UCI[4] repositories. Since all these datasets are originally labeled, we intentionally randomly sample 200 labeled instances and treat the rest of data as unlabeled to make a semi-supervised learning setting.

6.2 Experimental Results

Fig. 2 shows the test error v.s. the training size for different algorithms. The results clearly show that TSGS^{3}VM runs much faster than other methods. Specifically, Figs. 2d and 2h confirm the high efficiency of TSGS^{3}VM even on the datasets with one million samples. Besides, TSGS^{3}VM requires low memory benefiting from pseudo-randomness for generating random features, while BLS^{3}VM and S^{3}VM^{light} would be often out of memory on large scale datasets.

Fig. 3 shows the test error of different methods. The results were obtained at the optimal hyper-parameters for different algorithms. From the figure, it is clear that TSGS^{3}VM achieves similar generalization performance as that of BLS^{3}VM, S^{3}VM^{light}, and BGS^{3}VM methods which confirm that TSGS^{3}VM converge well in practice. Besides, TSGS^{3}VM achieves better generalization performance than FRS^{3}VM, because TSGS^{3}VM has the advantage that it would automatically use more and more random features (for each data x) as the number of iterations increases.

Based on these results, we conclude that TSGS^{3}VM is much more efficient and scalable than these algorithms while retaining the similar generalization performance.

7 Conclusion

In this paper, we provide a novel triply stochastic gradients algorithm for kernel S^{3}VM to make it scalable. We establish new theoretic analysis for TSGS^{3}VM which guarantees that TSGS^{3}VM can efficiently converge to a stationary point for a general non-convex learning problem under weak assumptions. As far as we know, TSGS^{3}VM is the first work that offers non-convex analysis for DSG-like algorithm without a strong initialization assumption. Extensive experimental results on a variety of benchmark datasets demonstrate the superiority of our proposed TSGS^{3}VM.

Acknowledgments

H.H. was partially supported by U.S. NSF IIS 1836945, IIS 1836938, DBI 1836866, IIS 1845666, IIS 1852606, IIS 1838627, IIS 1837956. B.G. was partially supported by the National Natural Science Foundation of China (No: 61573191), and the Natural Science Foundation (No. BK20161534), Six talent peaks project (No. XYDXX-042) in Jiangsu Province.

[2] BLS^{3}VM and S^{3}VM^{light} can be found in http://pages.cs.wisc.edu/ jerryzhu/ssl/software.html

[3] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

[4] http://archive.ics.uci.edu/ml/datasets.html

References

[Bennett and Demiriz, 1999] Kristin P Bennett and Ayhan Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information processing systems*, pages 368–374, 1999.

[Cai and Cherkassky, 2012] Feng Cai and Vladimir Cherkassky. Generalized smo algorithm for svm-based multitask learning. *IEEE transactions on neural networks and learning systems*, 23(6):997–1003, 2012.

[Carratino *et al.*, 2018] Luigi Carratino, Alessandro Rudi, and Lorenzo Rosasco. Learning with sgd and random features. In *Advances in Neural Information Processing Systems*, pages 10213–10224, 2018.

[Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[Chapelle and Zien, 2005] Olivier Chapelle and Alexander Zien. Semi-supervised classification by low density separation. In *AISTATS*, volume 2005, pages 57–64. Citeseer, 2005.

[Chapelle, 2007] Olivier Chapelle. Training a support vector machine in the primal. *Neural computation*, 19(5):1155–1178, 2007.

[Collobert *et al.*, 2006] Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. Large scale transductive svms. *Journal of Machine Learning Research*, 7(Aug):1687–1712, 2006.

[Dai *et al.*, 2014] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina F Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 3041–3049, 2014.

[Drineas and Mahoney, 2005] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *journal of machine learning research*, 6(Dec):2153–2175, 2005.

[Ghadimi and Lan, 2013] Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[Gu *et al.*, 2018a] Bin Gu, Zhouyuan Huo, Cheng Deng, and Heng Huang. Faster derivative-free stochastic algorithm for shared memory machines. In *International Conference on Machine Learning*, pages 1807–1816, 2018.

[Gu *et al.*, 2018b] Bin Gu, Yingying Shan, Xiang Geng, and Guansheng Zheng. Accelerated asynchronous greedy coordinate descent algorithm for svms. In *IJCAI*, pages 2170–2176, 2018.

[Gu *et al.*, 2018c] Bin Gu, Miao Xin, Zhouyuan Huo, and Heng Huang. Asynchronous doubly stochastic sparse kernel learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[Gu *et al.*, 2018d] Bin Gu, Xiao-Tong Yuan, Songcan Chen, and Heng Huang. New incremental learning algorithm for semi-supervised support vector machine. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1475–1484. ACM, 2018.

[Huo *et al.*, 2018] Zhouyuan Huo, Bin Gu, and Heng Huang. Training neural networks using features replay. In *Advances in Neural Information Processing Systems*, pages 6659–6668, 2018.

[Joachims, 1999] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Icml*, volume 99, pages 200–209, 1999.

[Le *et al.*, 2016] Trung Le, Phuong Duong, Mi Dinh, Tu Dinh Nguyen, Vu Nguyen, and Dinh Q Phung. Budgeted semi-supervised support vector machine. In *UAI*, 2016.

[Li *et al.*, 2017] Xiang Li, Bin Gu, Shuang Ao, Huaimin Wang, and Charles X Ling. Triply stochastic gradients on multiple kernel learning. In *UAI*, 2017.

[Lopez-Paz *et al.*, 2014] David Lopez-Paz, Suvrit Sra, Alex Smola, Zoubin Ghahramani, and Bernhard Schölkopf. Randomized nonlinear component analysis. *Computer Science*, 4:1359–1367, 2014.

[Rahimi and Recht, 2008] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[Wang *et al.*, 2007] Junhui Wang, Xiaotong Shen, and Wei Pan. On transductive support vector machines. *Contemporary Mathematics*, 443:7–20, 2007.

[Wendland, 2004] Holger Wendland. *Scattered data approximation*, volume 17. Cambridge university press, 2004.

[Xie *et al.*, 2015] Bo Xie, Yingyu Liang, and Le Song. Scale up nonlinear component analysis with doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 2341–2349, 2015.

[Yuille and Rangarajan, 2002] Alan L Yuille and Anand Rangarajan. The concave-convex procedure (cccp). In *Advances in neural information processing systems*, pages 1033–1040, 2002.