Utilize Old Coordinates: Faster Doubly Stochastic Gradients for Kernel Methods

Chun-Liang Li
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
chunlial@cs.cmu.edu

Barnabás Póczos
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
bapoczos@cs.cmu.edu

Abstract

To address the scalability issue of kernel methods, random features are commonly used for kernel approximation (Rahimi and Recht, 2007). They map the input data to a randomized low-dimensional feature space and apply fast linear learning algorithms on it. However, to achieve high precision results, one might still need a large number of random features, which is infeasible in large-scale applications. Dai et al. (2014) address this issue by recomputing the random features of small batches in each iteration instead of pre-generating for the whole dataset and keeping them in the memory. The algorithm increases the number of random features linearly with iterations, which can reduce the approximation error to arbitrarily small. A drawback of this approach is that the large number of random features slows down the prediction and gradient evaluation after several iterations. We propose two algorithms to remedy this situation by "utilizing" old random features instead of adding new features in certain iterations. By checking the expected descent amount, the proposed algorithm selects "important" old features to update. The resulting procedure is surprisingly simple without enhancing the complexity of the original algorithm but effective in practice. We conduct empirical studies on both medium and large-scale datasets, such as ImageNet, to demonstrate the power of the proposed algorithms.

1 INTRODUCTION

Kernel methods are powerful tools for learning non-linear hypotheses. Many algorithms can be combined with kernel methods, including SVM and logistic regression. However, kernel methods are usually considered as non-scalable due to the kernel matrix $K \in \mathbb{R}^{n \times n}$, where n is the number of examples. For large-scale datasets, such as MNIST-8M (8.1 million) and ImageNet (1.3 million), the memory usage makes kernels methods prohibitive for these applications.

One line of research is devoted to kernel approximation with limited memory usage (Williams and Seeger, 2000; Rahimi and Recht, 2007). Random Features (Rahimi and Recht, 2007), inspired by Bochner's theory, approximate the kernel mapping via a simple sampling procedure. After mapping the input data into the randomized feature space created by random features, we then apply existing fast linear learning algorithms. It has attracted machine learning community's interest because of its simplicity and effectiveness in practice. The extensions of random features include Rahimi and Recht (2008); Yang et al. (2012); Le et al. (2013); Yang et al. (2014); Chen et al. (2015); Bach (2015). However, both theoretical (Rahimi and Recht, 2007) and empirical studies show one might still need a large number of random features to achieve high precision results. If we pre-generate the random features and keep them in the memory, it is still infeasible in modern large-scale applications.

Dai et al. (2014) propose a remedy which generates random features on the fly by connecting functional gradients and random features, which is called *Doubly Stochastic Gradient Descent* (DSG). DSG re-computes the random features for a small batch of data in each iteration instead of keeping them in the memory. The re-computing manner allows DSG to increase the number of random features in every iteration. The algorithm can also be treated as a variant of random coordinate gradient (RCD) algorithm, because DSG updates the newly increased feature (coordinate) in every iteration. Therefore, by increasing the number of features, DSG can achieve arbitrarily small approximation error if one have sufficient budget of time. On the flip side, increasing features in every iteration causes the number of random features to grow linearly with number iterations. After several iterations, the large number of used random features makes prediction and computing gradients slow.

In this paper, we make the following contributions. First, we solve the drawback of large number of random features by updating old coordinates instead of increasing features in certain iterations (Section 4) to reduce the number of the used random features. One simple extension is sampling from old coordinates uniformly and periodically as typical RCD over the finite dimensions, which is called *DSG with Uniform Sampling* (UDSG). We make the second contribution by analyzing the convergence rate of UDSG. Although UDSG usually works well in practice, our theoretical analysis suggests the descent amount of UDSG is not as much as Dai et al. (2014) in the worst case. We then make our third contribution by proposing the other non-trivial algorithm, *DSG with Checking* (CDSG), which chooses the old coordinate by checking the expected descent amount and derive a expected line search methodology to further boost the performance. Theoretically, CDSG enjoys the same convergence rate as DSG. Empirically, it outperforms other algorithms. Last, we conduct experiments on large-scale datasets, including ImageNet, and further study the comparison with deep neural nets in Section 5. We then conclude in Section 6 and make a guideline of choosing algorithms in practice.

2 PRELIMINARIES

The problem we are interested in this paper is as follows. Assume the data point $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ is an *i.i.d.* sample from a distribution $\mathbb{P}(\mathbf{x}, y)$, where $\mathcal{X} \subseteq \mathbb{R}^d$, we want to estimate a function $f : \mathcal{X} \to \mathcal{Y}$ in Reproducing Hilbert Space (RKHS) $\mathcal{H}$ by optimizing

$$f_* = \operatorname*{argmin}_{f \in \mathcal{H}} R(f) = \operatorname*{argmin}_{f \in \mathcal{H}} \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2 + \mathbb{E}_{(\mathbf{x},y)}[\ell(f(\mathbf{x}), y)], \tag{1}$$

where $\ell(z, y)$ is a convex loss function in z. Commonly used loss functions are hinge loss (SVM), logistic loss (logistic regression) and square loss (ridge regression). Note that if the data comes from a batch setting, we replace the above expectation with the empirical expectation (average).

2.1 RKHS AND KERNEL

The RKHS $\mathcal{H}$ on $\mathcal{X}$ is a Hilbert space of functions from $\mathcal{X}$ to $\mathbb{R}$. One typical way to define RKHS is via kernel functions $k(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, which encodes the similarity between two data points. The kernel function k is symmetric and positive definite. $\mathcal{H}$ is RKHS if and only if there exists a kernel $k(\mathbf{x}, \mathbf{x}')$ such that $\forall \mathbf{x} \in \mathcal{X}, k(\mathbf{x}, \cdot) \in \mathcal{H}$ and $\langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot)\rangle_{\mathcal{H}} = k(\mathbf{x}, \mathbf{x}')$. Also, if $f \in \mathcal{H}$, $f(\mathbf{x}) = \langle f, k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}}$. One commonly used kernel is Gaussian RBF kernel $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$. The training bottleneck of using kernels is to compute and store the kernel matrix $K \in \mathbb{R}^{n \times n}$ for n data points. It brings the computational concern in both time and space complexity and makes designing scalable algorithms for large-scale problems a challenging task. There are several approach to addressing this difficulty by making trade-off between and time as space. For example, LIBSVM (Chang and Lin, 2011) only caches some columns of kernel matrix to save the memory usage and re-compute the columns where there is a cache miss.

2.2 RANDOM FEATURE

The other way to define kernel is finding the explicit feature mapping $\phi(\mathbf{x})$ such that $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$. Bochner's theorem suggests a way to find this mapping for the stationary (shift-invariant) kernel, *i.e.,* $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$, and draws the community's attention in this decade (Rahimi and Recht, 2007).

Theorem 1. *(Bochner's Theorem) A continuous, real-valued, symmetric and shift-invariant function k on $\mathbb{R}^d$ is a positive definite kernel if and only if there is a positive finite measure $\mathbb{P}(\boldsymbol{\omega})$ such that*

$$\begin{aligned} k(\mathbf{x} - \mathbf{x}') \quad = \quad & \textstyle\int_{\mathbb{R}^d} 2\left(\cos(\boldsymbol{\omega}^\top \mathbf{x}) \cos(\boldsymbol{\omega}^\top \mathbf{x}')\right. \\ & \left. + \sin(\boldsymbol{\omega}^\top \mathbf{x}) \sin(\boldsymbol{\omega}^\top \mathbf{x}')\right) d\mathbb{P}(\omega), \end{aligned}$$

For Gaussian RBF kernel, the corresponding density $\mathbb{P}(\omega)$ is the Gaussian distribution.

Inspired from Bochner's Theorem, we can approximate the kernel evaluation by the Monte-Carlo approximation. Define $\phi_\omega(\mathbf{x}) = \sqrt{2}[\cos(\boldsymbol{\omega}^\top \mathbf{x}), \sin(\boldsymbol{\omega}^\top \mathbf{x})]$ and $z(\mathbf{x}) = \frac{1}{\sqrt{m}}[\phi_{\boldsymbol{\omega}_1}(\mathbf{x}), \cdots, \phi_{\boldsymbol{\omega}_m}(\mathbf{x})]$, where $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_m$ are i.i.d. samples from $\mathbb{P}(\boldsymbol{\omega})$. We then have

$$k(\mathbf{x} - \mathbf{x}') = \mathbb{E}_\omega \left(\phi_\omega(\mathbf{x})\phi_\omega(\mathbf{x}')\right) \approx z(\mathbf{x})^\top z(\mathbf{x}')$$

and (1) can be transformed to

$$\mathbf{w}_* = \operatorname*{argmin}_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \mathbb{E}_{(\mathbf{x},y)}[l(\mathbf{w}^\top z(\mathbf{x}), y)], \tag{2}$$

which can be solved by fast linear learning algorithms to handle million-scale data easily (Fan et al., 2008).

3 DOUBLY STOCHASTIC KERNEL MACHINE

By using random features, we can represent the feature mapping with a finite-length vector $z(\mathbf{x})$ and approximate the kernel evaluation by the inner product $k(\mathbf{x}, \mathbf{x}') \approx z(\mathbf{x})^\top z(\mathbf{x}')$. We are then able to transform problem (1) into the linear learning problem (2). The approximation error $\mathbb{E}[(f_*(\mathbf{x}) - \mathbf{w}_*^\top(\mathbf{x}))^2] \le \epsilon$ can be bounded by ϵ with $O(1/\epsilon)$ number of random features (Rahimi and Recht, 2007), which is the consequence of the bound of Monte-Carlo approximation. More discussions on the optimality can be referred to Sriperumbudur and Szabó (2015).

Suppose we use m random features for approximation, then the space complexity for storing the transformed data is $O(nm)$, where n is the number of data points. One line of research is to replace the simple Monte-Carlo with different sampling scheme to improve the learning with random features. Le et al. (2013) propose an efficient sampling procedure to save the feature generation time. Yang et al. (2014); Chen et al. (2015); Bach (2015) use different sampling strategies with faster convergence rate to reduce m.

In practice, to achieve accurate results, m usually has to be large, which cause the space cost for storing. For example, MNIST-8M data contains eight million training points. If we use 10^5 random features with `double` type, it takes more than 1T memory to store the transformed data, which is prohibitive to many machines. On the other hand, the kernel matrix K is approximated by $z(\mathbf{X})z(\mathbf{X})^\top$, where $z(\mathbf{X}) \in \mathbb{R}^{n\times m}$. However, K is not low rank and has long-tailed eigenvalue distributions for many commonly used kernels (Weyl, 1912; Wathen and Zhu, 2015), which implies there is no hope to accurately approximate K with small m.

Therefore, the other line of research is to save memory usage with large m. Yen et al. (2014) propose a algorithm by imposing a sparsity constraint in (2) to reduce the memory usage. However, the sparsity constraint makes the problem not an unbiased approximation of (1). Dai et al. (2014) propose to re-generate the random features repeatedly during the training to save the memory usage, which makes using a large number of random features possible.

3.1 DOUBLY STOCHASTIC GRADIENT

Dai et al. (2014) propose a novel algorithm by using "doubly functional gradient" to address the scalability issue of kernel methods. Given a data point $(\mathbf{x}, y)$, the stochastic functional gradient of (1) is $\nabla_f R(f) = \lambda f(\cdot) + \mathbb{E}_{(\mathbf{x},y)}[\xi(\cdot)]$, where $\xi(\cdot) = \ell'(f(\mathbf{x}), y)k(\mathbf{x}, \cdot)$. By Bocher's theorem, we could further approximate the functional gradient $\xi(\cdot)$ by the random feature $\phi_{\boldsymbol{\omega}}(\mathbf{x})$. That is, given $\boldsymbol{\omega} \sim \mathbb{P}(\boldsymbol{\omega})$ and a data point $(\mathbf{x}, y)$, the doubly stochastic gradient of $\ell(f(\mathbf{x}), y)$ with respect to $f \in \mathcal{H}$ is $\zeta(\cdot) = l'(f(\mathbf{x}), y)\phi_{\boldsymbol{\omega}}(\mathbf{x})\phi_{\boldsymbol{\omega}}(\cdot)$, where $\mathbb{E}_{\boldsymbol{\omega}}(\zeta(\cdot)) = \xi(\cdot)$. The following formula connects the randomness from data $(\mathbf{x}, y)$ and random feature $\boldsymbol{\omega}$,

$$\begin{aligned}\nabla_f R(f) &= \lambda f(\cdot) + \mathbb{E}_{(\mathbf{x},y)}[\xi(\cdot)] \\ &= \lambda f(\cdot) + \mathbb{E}_{(\mathbf{x},y)}\mathbb{E}_{\boldsymbol{\omega}}[\zeta(\cdot)].\end{aligned}$$

In each iteration t, the doubly stochastic gradient of $\ell(f(\mathbf{x}_t), y_t)$ is $\zeta_t(\cdot) = \ell'(f_t(\mathbf{x}_t), y_t)\phi_{\boldsymbol{\omega}_t}(\mathbf{x}_t)\phi_{\boldsymbol{\omega}_t}(\cdot)$ by sampling $\boldsymbol{\omega}_t$ from $\mathbb{P}(\boldsymbol{\omega})$ and $(\mathbf{x}_t, y_t)$ from $\mathbb{P}(\mathbf{x}, y)$. The update rule for the algorithm is

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

where a_t^i are coefficients from λ and γ_t in each iteration. For each data point $\mathbf{x}$, we do not need to pre-generate the corresponding $z(\mathbf{x})$ until we want to evaluate the function as well as the doubly stochastic gradient on $\mathbf{x}$. The potential problem is how to regenerate the $z(\mathbf{x})$ every time. If we store $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_t$, which takes $O(dt)$, when both d and t are large, it is still infeasible in practice. Thanks for the pseudo-randomness used in modern computers, we could use different random seeds for different iterations, then we are guaranteed to sample the same $\boldsymbol{\omega}$ back. Then the space complexity is only $O(t)$ for storing $\alpha_t^i = a_t^i \ell'(f_i(\mathbf{x}_i), y_i)\phi_{\boldsymbol{\omega}}(\mathbf{x}_i)$. We call the algorithm as DSG, which is shown in Algorithm 1. The convergence rate of DSG is proved in Dai et al. (2014) under the conditions of Assumption 2.

Algorithm 1 $\{\alpha_i\}_{i=1}^t =$ **DSG**$(\mathbb{P}(\mathbf{x}, y))$

```
for $i = 1, \ldots, t$ do
    Sample $(\mathbf{x}_i, y_i) \sim \mathbb{P}(\mathbf{x}, y)$.
    Sample $\boldsymbol{\omega}_i \sim \mathbb{P}(\boldsymbol{\omega})$ with seed $i$.
    $f(\mathbf{x}_i) =$ Predict$(\mathbf{x}_i, \{\alpha_j\}_{j=1}^{i-1})$.
    $\alpha_i = -\gamma_i l'(f(\mathbf{x}_i), y_i)\phi_{\boldsymbol{\omega}_i}(\mathbf{x}_i)$.
    $\alpha_j = (1 - \gamma_i\lambda)\alpha_j$ for $j = 1, \ldots, i-1$.
end for
```

Algorithm 2 $f(\mathbf{x}) =$ **Predict**$(\mathbf{x}, \{\alpha_i\}_{i=1}^m)$

```
Set $f(\mathbf{x}) = 0$.
for $i = 1, \ldots, m$ do
    Sample $\boldsymbol{\omega}_i \sim \mathbb{P}(\boldsymbol{\omega})$ with seed $i$.
    $f(\mathbf{x}) = f(\mathbf{x}) + \alpha_i\phi_{\boldsymbol{\omega}_i}(\mathbf{x})$.
end for
```

Assumption 2.

1. *The optimal solution f_* to the problem (1) exists.*
2. *Loss function $\ell(u, y) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and its first-order derivative is L-Lipschitz continous in terms of the first argument.*
3. *There exists $M > 0$, such that $|\ell'(f_t(\mathbf{x}_t), y_t)| \leqslant M$. Note that in our situation $M < \infty$ exists since we assume bounded domain and the functions f_t we generate are always bounded as well.*
4. *There exists $\kappa > 0$ and $\phi > 0$, such that $k(\mathbf{x}, \mathbf{x}') \leqslant \kappa$, $|\phi_\omega(\mathbf{x})\phi_\omega(\mathbf{x}')| \leqslant \phi, \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \omega \in \Omega$. For Gaussian RBF kernel, we have $\kappa = 1$, $\phi = 2$.*

Theorem 3 (Convergence rate of DSG (Dai et al., 2014)). *When $\gamma_t = \frac{\theta}{t}$ with $\theta > 0$ such that $\theta\lambda \in (1, 2) \cup \mathbb{Z}_+$, for any $x \in \mathcal{X}$,*

$$\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t}\left[|f_{t+1}(x) - f_*(x)|^2\right] \leqslant \frac{2C_0^2 + 2\kappa S_0^2}{t},$$

where

$$S_0 = \max\left\{\|f_*\|_{\mathcal{H}}, \frac{Q_0 + \sqrt{Q_0^2 + Z(1+\theta\lambda)^2\theta^2\kappa M^2}}{Z}\right\},$$

with $Z = 2\lambda\theta - 1$*,* $Q_0 = \sqrt{2}\kappa^{1/2}LC_0\theta$*, and* $C_0 = 2(\kappa + \phi)M\theta$*.*

4 FAST DOUBLY STOCHASTIC KERNEL MACHINES

The $O(1/t)$ convergence rate in Theorem 3 is already optimal as proved in Dai et al. (2014). However, we could still improve DSG. In in each iteration, DSG is required to evaluate $f(\mathbf{x})$ for calculating gradients in each iteration, which needs to go through $\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_t$ to compute $\phi_{\boldsymbol{\omega}_i}(\mathbf{x})$ in Algorithm 2. For continuous distributions, such as Gaussian distribution $\mathbb{P}(\boldsymbol{\omega})$ for Gaussian kernel, the collision probability of sampling is zero. After t iterations, we have t different $\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_t$ for the feature generation. Then the complexity of the t^{th} iteration is $O(td)$, and the total complexity from iteration 1 to iteration t is $O(t^2 d)$, which makes DSG less efficient when t is large.

On the other hand, DSG could be treated as a random coordinate descent (RCD) with mini-batch. In each iteration t, sampling $\boldsymbol{\omega}_t$ can be treated as choosing one coordinate to update. Since the collision probability is zero, it is equivalent to updating a new coordinate in every iteration. From the perspective of RCD, we could also choose an "old" coordinate to update instead of sampling new $\boldsymbol{\omega}$ in certain iterations. Then we could reduce the number of used coordinates.

In what follows, we will present two algorithms, UDSG and CDSG, which "utilize" previous coordinates. UDSG is a nature extension from RCD. Studying its theoretical analysis gives us deeper insight to design the non-trivial algorithm CDSG, which enjoys the better bound of sample complexity than UDSG. In some following descriptions, we will abuse $\boldsymbol{\omega}_i$ as the coordinate index for convenience.

4.1 UTILIZE WITH UNIFORM SAMPLING

A simple strategy is using empirical distribution to approximate $\mathbb{P}(\boldsymbol{\omega})$, which is a widely-used technique in statistics. Assume we already independently sample $\boldsymbol{\Omega}^m = \{\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_m\}$ from $\mathbb{P}(\boldsymbol{\omega})$, then we could create an empirical distribution $\hat{\mathbb{P}}_m(\boldsymbol{\omega})$, which is a uniform distribution on $\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_m$. Sampling from $\hat{\mathbb{P}}_m(\boldsymbol{\omega})$ is unbiased since $\mathbb{E}_{\boldsymbol{\Omega}^m}\mathbb{E}_{\boldsymbol{\omega}\sim\hat{\mathbb{P}}_m}[\boldsymbol{\omega}|\boldsymbol{\Omega}^m] = \mathbb{E}_{\boldsymbol{\omega}\sim\mathbb{P}}(\boldsymbol{\omega})$. When m is large, we could expect the empirical distribution $\hat{\mathbb{P}}_m(\boldsymbol{\omega})$ to be a good approximation of $\mathbb{P}(\boldsymbol{\omega})$. Note that it is similar to typical RCD algorithm over a finite number of dimensions.

Here we study one simple algorithm for a concise presentation, which periodically samples from $\mathbb{P}(\boldsymbol{\omega})$ for G iterations and then sample from the empirical distribution $\hat{\mathbb{P}}(\boldsymbol{\omega})$ for U iterations. The number of random features m after t iterations is $O(Gt/G + U)$. The proposed algorithm, doubly stochastic gradients descent with uniform sampling, which is called UDSG and shown in Algorithm 3. Based on Assumption 2, the convergence rate of UDSG is shown in Theorem 4.

Algorithm 3 $\{\alpha_i\}_{i=1}^m$ = **UDSG**$(\mathbb{P}(\mathbf{x}, y))$

```
m = 0
for i = 1, ..., t do
  Sample (x_i, y_i) ~ P(x, y).
  f(x_i) = Predict(x_i, {α_j}_{j=1}^m).
  if mod(i, G + U) < G then
    Sample ω_{m+1} ~ P(ω) with seed m + 1.
    α_j = (1 − γ_i λ) α_j for j = 1, ..., m.
    α_{m+1} = −γ_i ℓ'(f(x_i), y_i) φ_{ω_{m+1}}(x_i).
    m = m + 1
  else
    Sample ω_k ~ P̂_m(ω), where 1 ≤ k ≤ m.
    α_j = (1 − γ_i λ) α_j for j = 1, ..., m.
    α_k = α_k − γ_i ℓ'(f(x_i), y_i) φ_{ω_k}(x_i).
  end if
end for
```

Theorem 4 (Convergence rate of UDSG). *When* $\gamma_t = \frac{\theta}{t}$ *with* $\theta > 0$ *such that* $\theta\lambda \in (1, 2) \cup \mathbb{Z}_+$*, for any* $\mathbf{x} \in \mathcal{X}$*,*

$$\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t}\left[|f_{t+1}(\mathbf{x}) - f_*(\mathbf{x})|^2\right] \leqslant \frac{2C_1^2 + 2\kappa S_1^2}{t},$$

where

$$S_1 = \max\left\{\|f_*\|_{\mathcal{H}}, \frac{Q_1 + \sqrt{Q_1^2 + Z(1+\theta\lambda)^2\theta^2\kappa M^2}}{Z}\right\},$$

with $Z = 2\lambda\theta - 1$*,* $Q_1 = \sqrt{2}\kappa^{1/2}LC_1\theta$*, and* $C_1 = 2(\kappa + \phi)M\theta\sqrt{1 + \frac{2U}{G+U} + \frac{2U^2}{G(G+U)} + \frac{2U(U-1)}{Gt}}$

4.1.1 PROOF OF THEOREM 4

We provide a proof sketch here. Our analysis closely follows Dai et al. (2014) but we need to carefully deal with several cross-terms as shown in (3) later. Some technical lemmas are in Appendix.

We decompose the error into two terms,

$$|f_t(\mathbf{x}) - f^*(\mathbf{x})|^2 \le 2|f_t(\mathbf{x}) - h_t(\mathbf{x})|^2 + 2\kappa\|h_t - f^*\|_{\mathcal{H}}^2,$$

where

$$h_t(\cdot) = \mathbb{E}_{\boldsymbol{\omega}}[f_t(\cdot)] = \mathbb{E}_{\boldsymbol{\omega}}[\sum_{i=1}^t a_t^i \zeta_{\boldsymbol{\omega}_i}(\cdot)] = \sum_{i=1}^t a_t^i \xi_{\boldsymbol{\omega}_i}(\cdot).$$

Since $\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t}\left[\|h_{t+1} - f_*\|_{\mathcal{H}}^2\right] \leqslant \frac{S^2}{t}$, where S is a constant related to $\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t}\left(|f_{t+1}(\mathbf{x}) - h_{t+1}(\mathbf{x})|^2\right)$ as shown in Dai et al. (2014), the remaining task is to bound $\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t}\left(|f_{t+1}(\mathbf{x}) - h_{t+1}(\mathbf{x})|^2\right)$. We define the following terms to simplify the notations.

- $\mathcal{G} = \{x | (x-1) \bmod (G+U) < G \text{ and } 1 \le x \le t\}$.
- $\mathcal{G}_k = \{x | x \in \mathcal{G} \text{ and } \lceil x/(G+U) \rceil = k\}$.
- $\mathcal{U}_k = \{x | x \notin \mathcal{G}, \lceil x/(G+U) \rceil = k \text{ and } 1 \le x \le t\}$.
- $V_i(\mathbf{x}) = a_t^i \left(\zeta_t(\mathbf{x}) - \xi_t(\mathbf{x})\right)$.

By definition, we have $\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t} \left(|f_{t+1}(\mathbf{x}) - h_{t+1}(\mathbf{x})|^2 \right) = \mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t} \left[\left(\sum_{i=1}^t V_i(\mathbf{x}) \right)^2 \right]$, then

$$\begin{aligned}
& \mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t} \left[\left(\textstyle\sum_{i=1}^t V_i(\mathbf{x}) \right)^2 \right] \\
= \; & \mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t} \left[\left(\sum_{\mathcal{G}} V_i(\mathbf{x}) + \sum_{k=1}^{\lceil t/(G+U) \rceil} \sum_{i \in \mathcal{U}_k} V_i(\mathbf{x}) \right)^2 \right] \\
\le \; & \mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t} \left[\left(\sum_{\mathcal{G}} V_i(\mathbf{x}) \right)^2 + \sum_{k=1}^{\lceil t/(G+U) \rceil} \left(\sum_{i \in \mathcal{U}_k} V_i(\mathbf{x}) \right)^2 \right. \\
& + 2 \sum_{k=1}^{\lceil t/(G+U) \rceil} \left(\sum_{i \in \mathcal{U}_k} V_i(\mathbf{x}) \right) \left(\sum_{\mathcal{G}} V_i(\mathbf{x}) \right) \\
& \left. \sum_{p=1}^{\lceil \frac{t}{G+U} \rceil - 1} \sum_{q=p+1}^{\lceil \frac{t}{G+U} \rceil} \left(\sum_{i \in \mathcal{U}_p} V_i(\mathbf{x}) \right) \left(\sum_{i \in \mathcal{U}_q} V_i(\mathbf{x}) \right) \right].
\end{aligned} \tag{3}$$

We complete the proof by bounding each term. For details please refer to the Appendix.

4.1.2 TOTAL COMPLEXITY

The sample complexity bound in Theorem 4 is clearly worse than the result in Theorem 3 for DSG. However, we should take the complexity of single iteration into account for comparison. After t iterations, the ratio of the number of used $\boldsymbol{\omega}$ between UDSG and DSG is $O(G/G+U)$.

We suppose Q_1^2 in Theorem 4 dominates $Z(1+\theta\lambda)^2\theta^2\kappa M^2$. Also, we assume $S_0 > \|f_*\|_{\mathcal{H}}$ and $S_1 > \|f_*\|_{\mathcal{H}}$ in Theorem 3 and Theorem 4, respectively. The ratio between sample complexities of Theorem 3 and Theorem 4 is close to $O(1 + \frac{2U}{G+U} + \frac{2U^2}{G(G+U)} + \frac{2U(U-1)}{Gt})$. Then the ratio r between the total complexity of DSG and UDSG is

$$\begin{aligned}
r \; & \ge \; \left(1 + \tfrac{2U}{G+U} + \tfrac{2U^2}{G(G+U)}\right) \times \tfrac{G}{G+U} \\
& = \; \left(1 + \tfrac{2U}{G+U} \times \tfrac{G+U}{G}\right) \tfrac{G}{G+U} \\
& = \; \tfrac{G+2U}{G+U},
\end{aligned}$$

This result suggests setting U to be zero to minimize the complexity, which implies DSG is theoretically no worse than UDSG under certain conditions. However, this pessimistic result inspires us to design a better algorithm in Section 4.2.

4.2 UTILIZE WITH CHECKING

In each iteration t, we could either pick up the coordinate we have used, or sample a new coordinate from $\mathbb{P}(\boldsymbol{\omega})$ to update. The uniform sampling strategy algorithm, UDSG, has a worse bound than DSG, which implies updating a new coordinate $\boldsymbol{\omega} \sim \mathbb{P}(\boldsymbol{\omega})$ reduces the objective more than updating old coordinates in expectation. Therefore, before updating, we should "*check*" the old coordinate to ensure the chosen coordinate $\boldsymbol{\omega}_k$ is as effective as the newly sampled coordinate $\boldsymbol{\omega} \sim \mathbb{P}(\boldsymbol{\omega})$.

We start by investigating the reason that causes UDSG to be worse than DSG in expectation. We take the simple case that we sample $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_{t-1}$ from $\mathbb{P}(\boldsymbol{\omega})$ as DSG and set $\boldsymbol{\omega}_t = \boldsymbol{\omega}_k$, where $1 \le k \le t-1$. Then $\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t}(|f_{t+1}(\mathbf{x}) - h_{t+1}(\mathbf{x})|^2) = \mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t} \left[\left((\sum_{i=1, i \ne k}^{t-1} V_i(\mathbf{x})) + (V_k(\mathbf{x}) + V_t(\mathbf{x})) \right)^2 \right]$. We abbreviate $\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t}(\cdot)$ as $\mathbb{E}(\cdot)$ in the following context for a succinct representation.

Lemma 5. *(Dai et al., 2014)*

If $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}_j$ are i.i.d. samples from $\mathbb{P}(\omega)$, $\mathbb{E}\left[(V_i(\mathbf{x}) + V_j(\mathbf{x}))^2 \right] = \mathbb{E}\left(V_i(\mathbf{x})^2 \right) + \mathbb{E}\left(V_j(\mathbf{x})^2 \right)$.

We then have the bound

$$\begin{aligned}
& \mathbb{E}\left[\left(\textstyle\sum_{i=1}^t V_i(\mathbf{x}) \right)^2 \right] \\
\le \; & \sum_{i=1, i \ne k}^{t-1} \mathbb{E}\left(V_i(\mathbf{x})^2 \right) + \mathbb{E}\left[\left(|V_k(\mathbf{x})| + |V_t(\mathbf{x})| \right)^2 \right].
\end{aligned} \tag{4}$$

In contrast, the bound of DSG is

$$\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t} \left[\left(\sum_{i=1}^t V_i(\mathbf{x}) \right)^2 \right] \le \sum_{i=1}^t \mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t} \left(V_i(\mathbf{x})^2 \right). \tag{5}$$

The cross term $\mathbb{E}_{\mathcal{D}^t, \boldsymbol{\omega}^t}(|V_k(\mathbf{x})||V_t(\mathbf{x})|)$ from expanding the quadratic term in (4) causes a worse bound than (5). The generalization of this simple example is Theorem 4 in Section 4.1. The cross terms cause a larger constant C_1 in Theorem 4 than C_0 in Theorem 3.

Without any further knowledge about $V_k(\mathbf{x})$ and $V_t(\mathbf{x})$, the upper bound in (4) is unavoidable. Therefore, in iteration t, we should choose $1 \le k \le t-1$ such that the upper bound U_t of $\mathbb{E}\left[(V_k(\mathbf{x}) + V_t(\mathbf{x}))^2 \right]$ can be bounded by the bound of $\mathbb{E}\left(V_k(\mathbf{x})^2 \right) + \mathbb{E}\left(V_t(\mathbf{x})^2 \right)$. To be specific, we should select k such that

$$U_t \le \mathbb{E}\left[\left(a_t^k \ell'(f_k(\mathbf{x}_k), y_k) \right)^2 + \left(a_t^t \ell'(f_t(\mathbf{x}_t), y_t) \right)^2 \right] (\kappa + \phi)^2. \tag{6}$$

For convenience, we let $g_i = \ell'(f_i(\mathbf{x}_i), y_i)$ and $\Phi_{\mathbf{x}}(\boldsymbol{\omega}_i, \mathbf{x}_k) = \phi_{\boldsymbol{\omega}_i}(\mathbf{x}_k)\phi_{\boldsymbol{\omega}_k}(\mathbf{x}) - k(\mathbf{x}_k, \mathbf{x})$. Expanding

$V_k(\mathbf{x})$ and $V_t(\mathbf{x})$ results the upper bound U_t as

$$\begin{aligned} & \mathbb{E}\left[(V_k(\mathbf{x})+V_t(\mathbf{x}))^2\right] \\ = \ & \mathbb{E}\left[\left(a_t^k g_k \Phi_{\mathbf{x}}(\boldsymbol{\omega}_k,\mathbf{x}_k)+a_t^t g_t \Phi_{\mathbf{x}}(\boldsymbol{\omega}_t,\mathbf{x}_t)+\right.\right. \\ & \left.\left. a_t^t g_t \Phi_{\mathbf{x}}(\boldsymbol{\omega}_k,\mathbf{x}_k)-a_t^t g_t \Phi_{\mathbf{x}}(\boldsymbol{\omega}_k,\mathbf{x}_k)\right)^2\right] \\ \leq \ & 2\mathbb{E}\left[\left(a_t^k g_k + a_t^t g_t\right)^2\right](\kappa+\phi)^2 \\ & +2\left((a_t^t M)^2\right)\mathbb{E}\left[\left(\Phi_{\mathbf{x}}(\boldsymbol{\omega}_k,\mathbf{x}_t)-\Phi_{\mathbf{x}}(\boldsymbol{\omega}_k,\mathbf{x}_k)\right)^2\right]. \end{aligned} \tag{7}$$

Note that we suppose $\boldsymbol{\omega}_t = \boldsymbol{\omega}_k$.

If x_1 and x_2 are *i.i.d.* samples from $\mathbb{P}(x)$, then $\mathbb{E}\left((x_1-x_2)^2\right) = 2\text{Var}(x)$. Let $\sigma^2 = \mathbb{E}_{\boldsymbol{\omega}_k}\left[\text{Var}_y\left(\Phi_{\mathbf{x}}(\boldsymbol{\omega}_k,y)\right)\right]$ and $\beta_t^k = a_t^k g_k$. Incorporating (7) into (6) with the above fact of variance results the selection criterion as choosing $\boldsymbol{\omega}_k$ such that

$$\begin{aligned} & 2\mathbb{E}\left[(\beta_t^k + a_t^t g_t)^2\right](\kappa+\phi)^2 + 2\left(a_t^t M\sigma(\kappa+\phi)\right)^2 \\ \leq \ & \mathbb{E}\left[(\beta_t^k)^2+(a_t^t g_t)^2\right](\kappa+\phi)^2. \end{aligned} \tag{8}$$

4.2.1 MINI-BATCH AND EXPECTED LINE SEARCH

In (8), smaller $a_t^t M\sigma(\kappa+\phi)$ makes it easier to find an $\boldsymbol{\omega}_k$ which satisfies (8). Reducing $a_t^t M\sigma(\kappa+\phi)$ comes for free by using the mini-batch extension of stochastic gradient descent. If the batch size is B, the variance of $\frac{1}{B}\sum_{i=1}^B \Phi_{\mathbf{x}_i}(\boldsymbol{\omega}_k,y)$ is $\frac{1}{B}\text{Var}_y\left(\Phi_{\mathbf{x}}(\boldsymbol{\omega}_k,y)\right)$.

Also, instead of setting $a_t^t = -\gamma_t$, we could further do line-search in expectation for a steeper descent with a larger step size. Assume the step size is η, and we want to keep the error upper bounded by DSG. Therefore, given $\boldsymbol{\omega}_1,\ldots,\boldsymbol{\omega}_{t-1}$, combining with the mini-batch extension, We find the largest step size $\eta < -\gamma_t$, such that

$$2(\beta_t^k+\eta g_t)^2 + \frac{2\left(\eta M\sigma\right)^2}{B^2} \leq (\beta_t^k)^2 + (a_t^t g_t)^2. \tag{9}$$

The optimization problem (9) is a quadratic inequality, which can be solved with the analytical solution. We then choose $\boldsymbol{\omega}_k$ with the max step size $|\eta_k|$ to update. We call the algorithm as "doubly stochastic gradient descent with checking" (CDSG), which is shown in Algorithm 4. For simplicity, we show the case when $B=1$ in Algorithm 4, but one can extend it to general $B>1$ cases.

Convergence rate of CDSG. The convergence rate of CDSG is exactly the same as Theorem 3. We omit the proof in the Appendix.

5 EXPERIMENT

In this section, we first study medium-scale data, which allows us to do thorough comparisons to understand the trade-off between different algorithms. We then also show the results on the large-scale datasets. Last, we compare DSG-based algorithms with deep neural nets. The details of the datasets are shown in Table 1.

Algorithm 4 $\{\alpha_i\}_{i=1}^m$ = **CDSG**($\mathbb{P}(\mathbf{x},y)$)

```
m = 0
for $i = 1,\ldots,t$ do
    Sample $(\mathbf{x}_i, y_i) \sim \mathbb{P}(\mathbf{x},y)$.
    $f(\mathbf{x}_i) = $ Predict$(\mathbf{x}_i, \{\alpha_j\}_{j=1}^m)$.
    Compute step sizes $\eta_1,\ldots,\eta_m$ via (9), and suppose $|\eta_k| = \text{argmax}_j |\eta_j|$.
    if $|\eta_j| \leq \gamma_t$ then
        $\alpha_j = (1-\gamma_i\lambda)\alpha_j$ for $j = 1,\ldots,m$.
        $\alpha_{m+1} = -\gamma_i \ell'(f(\mathbf{x}_i),y_i)\phi_{\boldsymbol{\omega}_{m+1}}(\mathbf{x}_i)$.
        $\beta_j = (1-\gamma_i\lambda)\beta_j$ for $j = 1,\ldots,m$.
        $\beta_{m+1} = -\gamma_i \ell'(f(\mathbf{x}_i),y_i)$.
        $m = m+1$
    else
        $\alpha_j = (1-\eta_k\lambda)\alpha_j$ for $j = 1,\ldots,m$.
        $\alpha_k = \alpha_k - \eta_k \ell'(f(\mathbf{x}_i),y_i)\phi_{\boldsymbol{\omega}_k}(\mathbf{x}_i)$.
        $\beta_j = (1-\eta_k\lambda)\alpha_j$ for $j = 1,\ldots,m$.
        $\beta_k = \alpha_k - \eta_k \ell'(f(\mathbf{x}_i),y_i)$.
    end if
end for
```

Dataset	# train	# test	# classes
CIFAR-10	60K	10K	10
Epsilon	0.4M	0.1M	2
Year	0.46M	51K	$\mathbb{R}$
MNIST-8M	8.1 M	10K	10
ImageNet 2012	1.3 M	0.1M	1000

Table 1: The dataset information for experiments, where the label of Year is real number.

General Setting In all experiments, we use Gaussian kernel. The kernel bandwidth is obtained by the median trick (Smola, 2004). For UDSG, we set $G=1$ and $U=1$. The other parameters will be specified later.

5.1 MEDIUM-SCALE DATA

We observed that the algorithms tend to overfit the medium-scale data without carefully tunning regularization terms. In this subsection, we only compare the training objectives, which is a natural criterion for optimization problems.

Setting for Medium-Scale Data For the CIFAR-10, we use raw pixels as input. Also, we conduct PCA to reduce the number of dimension of every datasets to be no more than 100. The parameters for DSG-based algorithms are shown in Table 2, where the feature block F means we sample F random features in each iteration. For non-DSG-

Dataset	Batch Size B	Feature Block F	λ
CIFAR10	$4,096$	256	10^{-5}
Epsilon	$10,000$	512	10^{-5}
Year	$10,000$	512	10^{-5}

Table 2: Parameters for DSG-based algorithms on medium-scale datasets.

based algorithms, which pre-generate random features for the data points, we set the number of random features as $4F$.

Compare with Other Approximation There are several competitors that can be considered, such as Kivinen et al. (2004), kernel SDCA (Shalev-Shwartz and Zhang, 2013), and several stochastic optimization algorithms with random feature or Nyström method (Shalev-Shwartz and Zhang, 2013; Johnson and Zhang, 2013; Nesterov, 2012). Here we only choose SVRG (Johnson and Zhang, 2013) with random features, which has been shown to be one of the best among the above algorithms.

We compare DSG, UDSG, CDSG and SVRG (Johnson and Zhang, 2013) with multi-class kernel logistic regression and kernel ridge regression. The results are shown in Figure 1.

We first look at Figure 1(a), Figure 1(b) and Figure 1(c). Since we only use $4F$ random features for SVRG, is not surprising that SVRG with limited random features converges to unsatisfactory results at the early stage. Without the enough number of the random features, the bottleneck of the learning is the approximation error instead of the optimization algorithms. Although we can use more features than $4F$ to achieve better performance for medium-scale data, it takes longer time and more memory for training. This issue becomes more critical in the large-scale applications. The memory usage can even make this approach prohibitive to large-scale problems. Note that we do not show SVRG in Figure 1(d), Figure 1(e) and Figure 1(f) since it converges out of the range.

Second, we compare DSG-based algorithms. In Figure 1(b), UDSG performs worse than DSG, which confirms the bound of Theorem 4 and the worst-case discussion of uniform sampling in Section 4.2. However, we observe that the worst case does not occur often in practice. Although the analysis in Section 4.1.2 is pessimistic, UDSG generally outperforms DSG in Figure 1. On the other hand, the proposed CDSG outperforms both DSG and UDSG in all datasets, which justifies the correctness of the validity of the proposed checking rule and expected line search in Section 4.2.

Compare with Exact Optimization We compare DSG-based algorithms with LIBSVM (Chang and Lin, 2011), which is the state-of-the-art solver for kernel learning by using kernel matrices. We consider kernel SVM in this comparison since there is no kernel logistic regression implementation in LIBSVM. We report the performance of each algorithm when LIBSVM converges. CIFAR-10 and Epsilon take LIBSVM 583 and 8457 seconds to converge, respectively. The results are shown in Table 3.

	LIBSVM	DSG	UDSG	CDSG
CIFAR-10	0.386	0.407	0.402	0.402
Epsilon	0.143	0.145	0.146	0.145

Table 3: The training error of each algorithm at the time when LIBSVM converges.

In these experiments, to achieve high-precision results, the exact optimization is always preferable in practice if we have enough memory. We observe that in both CIFAR-10 and Epsilon, DSG-based algorithms achieve the satisfactory performance quickly, but they take longer time to converge to the precise results. For example, all DSG-based algorithms can achieve 0.15 error of Epsilon within 150 seconds. Especially, CDSG can achieve 0.147. However, they take more than $10,000$ seconds to get the error lower than 0.144. We address this issue to the variance from doubly stochastic methodology, which requires the learning rate to be small and make the learning slow in the later iterations as the typical SGD algorithms (Johnson and Zhang, 2013; Shalev-Shwartz and Zhang, 2013).

5.2 LARGE-SCALE DATA

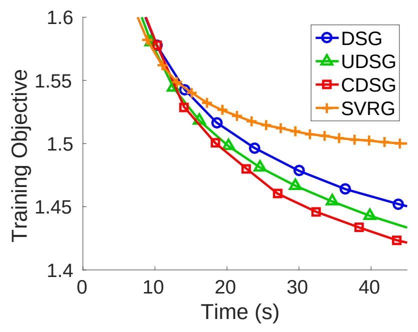
We study MNIST-8M digit recognition dataset, which contains 8.1 million training data and 10 classes. We reduce the raw pixel to 100 dimensions as feature and set $B = 20,000$, $F = 4096$ and $\lambda = 10^{-6}$. LIBSVM cannot run on this dataset due to the memory limitation, and we can only load 500 pre-generated random features into our memory, which results in unsatisfactory result (0.05 training and testing error) with any algorithms, such as SVRG. Therefore, we focus on the comparison on DSG-based algorithms. The result is shown in Figure 2[1].

As one can see from Figure 2, CDSG consistently outperforms the other two algorithms in both training objectives and testing error as well as under different loss functions. The results justify the correctness and usefulness of the proposed CDSG again. In contrast, UDSG does not perform as well as Figure 1, which confirms the worst case analysis in Theorem 4.

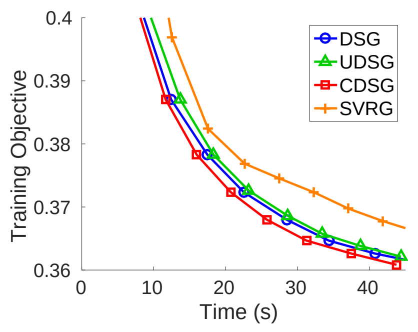
5.3 STUDY WITH DEEP NEURAL NET

We follow Dai et al. (2014) to compare DSG-based algorithms with deep neural nets on image classification

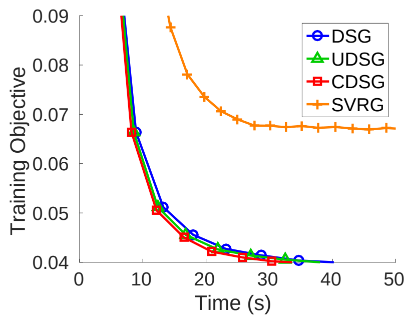
[1]Compared with multi-class logistic regression, the training objective of one-vs-one kernel SVM is less meaningful, so we do not compare it.



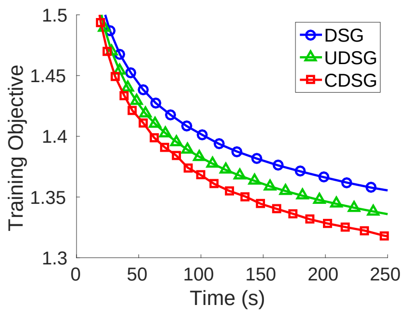
(a) CIFAR-10/Logistic Regression



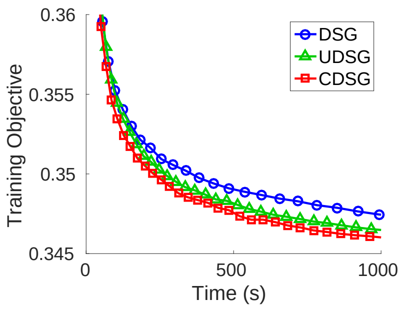
(b) Epsilon/Logistic Regression



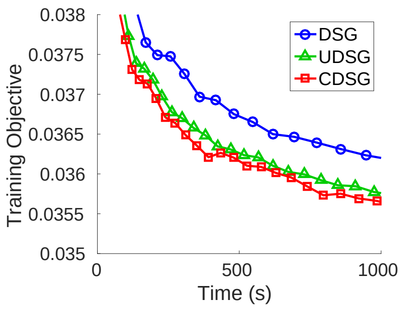
(c) Year/Ridge Regression



(d) CIFAR-10/Logistic Regression



(e) Epsilon/Logistic Regression



(f) Year/Ridge Regression

Figure 1: Training objective of different algorithms on each datasets.

Data	Batch Size b	Feature Block r	λ
CIFAR-10	32768	512	0.0005
MNIST-8M	16384	512	0.0005
ImageNet	16384	128	0.0005

Table 4: The parameters for DSG-based algorithms.

	DSG	UDSG	CDSG	Fixed	Joint
CIFAR-10	16.0	15.9	**15.8**	18.0	19.1
MNIST-8M	6.1	5.9	**5.3**	7.1	8.5
ImageNet	45.1	44.9	**44.7**	48.2	58.6

Table 5: The testing error (%) of each dataset when CDSG converges.

	DSG	UDSG	CDSG	Fixed	Joint
CIFAR-10	15.8	15.9	**15.8**	15.8	15.9
MNIST-8M	5.3	5.4	**5.3**	7.1	6.2
ImageNet	44.9	44.8	44.7	46.2	**42.4**

Table 6: The converged results (%) of all algorithms for each dataset.

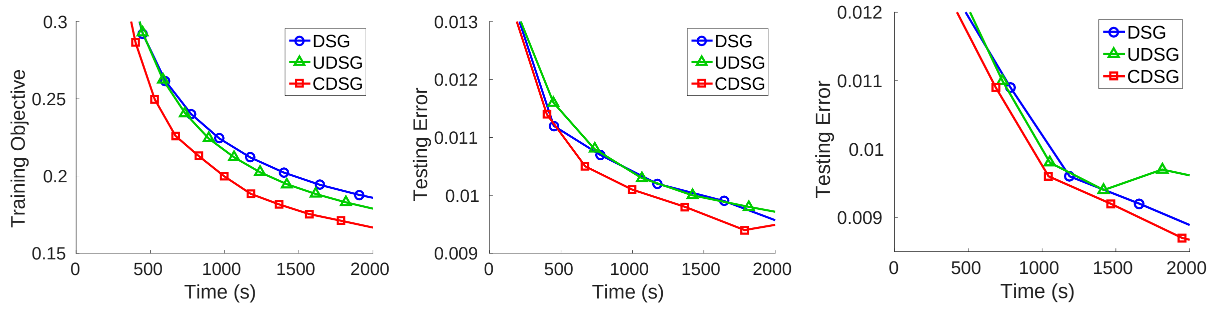
tasks. Besides CIFAR-10 and MNIST-8M, we also study ImageNet-2012, which is one of the most challenging image classification data currently. Since we do not aim to compare the representation learning ability, we use the pre-trained features provided by Dai et al. (2014). The detailed information of the used neural-net architectures (LeCun et al., 1998; Krizhevsky et al., 2012) can be found in the Appendix.

We study the following algorithms and the parameters are shown in Table 4.

- **Joint:** Put two fully connected layers at the top of the neural net for classification and train these two layers and the pre-trained layers "jointly".
- **Fixed:** Put two fully connected layers at the top of the neural net for classification and only train these two layers and without modifying pre-trained layers.
- **DSG-based:** Apply DSG algorithms on the pre-trained features.

Since the joint-trained neural net takes much more time than other algorithms, we report the results when CDSG converges in Table 5. The result shows that DSG-based algorithms converge faster than deep neural nets, and CDSG is the best in this family. For example, CDSG converges to 44.7% testing error on ImageNet. At the same time, the joint-trained neural net only achieves 58.6% testing error.

The converged results of all algorithms are shown in Table 6. In both CIFAR-10 and MNIST-8M, the CDSG algorithm is better than neural nets, which suggests proposed CDSG is not only efficient but also effective in some tasks. However, in ImageNet, the joint-trained neural net is still



(a) Training objective of logistic regression (b) Testing error of logistic regression (c) Testing error of SVM

Figure 2: Training objective of logistic regression on MNIST-8M.

the state-of-art by using more than twice of the training time needed for CDSG. We address the performance gap to the unsatisfactory representativeness of the pre-trained features, because the time for learning pre-trained features provided by Dai et al. (2014) only takes less than 10% of the training time. We could expect better pre-trained features could result in better performance of CDSG, but it takes more time for getting pre-trained features. Also, if we only have limited training time budget, such as the fine-tunning step in training deep neural networks, CDSG enjoys the advantage of the fast training to achieve satisfactory results quickly as shown in Table 5.

6 CONCLUSION

In this paper, we extended the DSG algorithm (Dai et al., 2014) to be more efficient by utilizing previous coordinates. We studied two algorithms including UDSG and CDSG. UDSG samples old coordinates with uniform sampling, which results in a worse convergence bound than the original DSG, though it outperforms DSG usually in practice. We also propose the other variant, CDSG, which selects previous coordinates in a more conservative way by checking the upper bound of the expected error. In the theoretical side, CDSG enjoys the same bound as DSG; in the practical side, CDSG is demonstrated to have better performance than DSG and UDSG consistently in all datasets we studied.

From our empirical study, we make the following suggestions.

Medium-Scale Data: If the size of memory permits and we want to achieve high-precision result, the solver with exact optimization by computing kernel matrices is still preferable, such as LIBSVM. However, they take longer time than the time needed for CDSG to achieve satisfactory performance (less than 5 minutes in all datasets we studied). For some medium-scale data we studied, the exact optimization solver takes hours to converge.

Large-Scale Data: For large-scale problems, such as ImageNet, the proposed CDSG algorithm enjoys several advantages. First, it is memory efficient, so we are allowed to use a large number of random features for kernel approximation, Second, it is computationally efficient and much faster than the original DSG algorithm. Under the situation with limited time budget, CDSG can quickly achieve satisfactory performance. However, if we want to achieve the state-of-the-art performance as deep neural nets, the proposed kernel approximation needs better feature representation as input to achieve better performance. Otherwise, the jointly-trained neural net may be preferable. The other alternative is combining the recent development of unsupervised training of deep neural networks (Doersch et al., 2015) with CDSG, which could possibly give us competitive performance with jointly-trained neural nets.

References

Bach, F. R. (2015). On the equivalence between quadrature rules and random features. *CoRR*.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*.

Chen, X., Yang, H., King, I., and Lyu, M. R. (2015). Training-efficient feature map for shift-invariant kernels. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Dai, B., Xie, B., He, N., Liang, Y., Raj, A., Balcan, M., and Song, L. (2014). Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*.

Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *International Conference on Computer Vision*.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*.

Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*.

Kivinen, J., Smola, A. J., and Williamson, R. C. (2004). Online Learning with Kernels. *IEEE Transactions on Signal Processing*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Le, Q. V., Sarlós, T., and Smola, A. J. (2013). Fastfood - computing hilbert space expansions in loglinear time. In *Proceedings of the International Conference on Machine Learning*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*.

Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. In *NIPS*.

Rahimi, A. and Recht, B. (2008). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems*.

Shalev-Shwartz, S. and Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*.

Smola, A. J. (2004). An introduction to machine learning with kernels lecture 5.

Sriperumbudur, B. K. and Szabó, Z. (2015). Optimal rates for random fourier features.

Wathen, A. J. and Zhu, S. (2015). On spectral distribution of kernel matrices related to radial basis functions. *Numerical Algorithms*.

Weyl, H. (1912). Das asymptotische verteilungsgesetz der eigenwerte linearer partieller differentialgleichungen (mit einer anwendung auf die theorie der hohlraumstrahlung). *Mathematische Annalen*.

Williams, C. K. I. and Seeger, M. W. (2000). Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*.

Yang, J., Sindhwani, V., Avron, H., and Mahoney, M. W. (2014). Quasi-monte carlo feature maps for shift-invariant kernels. In *Proceedings of the International Conference on Machine Learning*.

Yang, T., Li, Y.-F., Mahdavi, M., Jin, R., and Zhou, Z.-H. (2012). "nystr"om method vs random fourier features: A theoretical and empirical comparison". In *Advances in Neural Information Processing Systems*.

Yen, I. E., Lin, T., Lin, S., Ravikumar, P. K., and Dhillon, I. S. (2014). Sparse random feature algorithm as coordinate descent in hilbert space. In *Advances in Neural Information Processing Systems*.