Machine Learning (2022) 111:997–1018
https://doi.org/10.1007/s10994-021-06075-7

Improving kernel online learning with a snapshot memory

Trung Le[1] · **Khanh Nguyen**[2] · **Dinh Phung**[1,2]

Received: 19 May 2021 / Revised: 14 August 2021 / Accepted: 22 September 2021 /
Published online: 7 January 2022

Abstract
We propose in this paper the *Stochastic Variance-reduced Gradient Descent for Kernel Online Learning* (DualSVRG), which obtains the ϵ-approximate *linear* convergence rate and is not vulnerable to the curse of kernelization. Our approach uses a variance reduction technique to reduce the variance when estimating full gradient, and further exploits recent work in dual space gradient descent for online learning to achieve model optimality. This is achieved by introducing *the concept of an instant memory*, which is a snapshot storing the most recent incoming data instances and proposing *three transformer oracles*, namely budget, coverage, and always-move oracles. We further develop rigorous theoretical analysis to demonstrate that our proposed approach can obtain the ϵ-approximate *linear* convergence rate, while maintaining model sparsity, hence encourages fast training. We conduct extensive experiments on several benchmark datasets to compare our DualSVRG with state-of-the-art baselines in both batch and online settings. The experimental results show that our DualSVRG yields superior predictive performance, while spending comparable training time with baselines.

Keywords Kernel online learning · Incremental stochastic gradient descent · Online learning · Kernel methods · Stochastic optimization

Editors: Yu-Feng Li, Mehmet Gönen, Kee-Eung Kim.

✉ Trung Le
trunglm@monash.edu

Khanh Nguyen
nkhanh@vinai.io

Dinh Phung
dinh.phung@monash.edu

[1] Department of Data Science and AI, Monash University, Melbourne, Australia

[2] VinAI Research, Hanoi, Vietnam

1 Introduction

An optimization problem in machine learning is usually expressed as the sum of the average of the loss function over training data and a regularization term. Given this type of objective function, it is very computationally expensive to evaluate the full gradient needed in gradient descent, hence motivating techniques to estimate this quantity. Stochastic gradient descent (SGD) is such a technique that estimates the full gradient using the reduced gradient at a data instance randomly drawn from training set. The typical convergence rate for SGD up to the T-th iteration is known to be $\mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$ (Hazan et al. 2007; Shalev-Shwartz et al. 2007). Subsequently, several works have achieved a better convergence rate, i.e., $\mathcal{O}\left(\frac{1}{T}\right)$, for the strongly convex case (Rakhlin et al. 2012; Hazan and Kale 2014). Nonetheless, due to its high variance in estimating the full gradient, SGD-based method cannot achieve a higher convergence rate (e.g., the *linear* convergence rate), hence raising the necessity to devise estimators of the full gradient with lower variance. Recent incremental gradient methods (Schmidt et al. 2013; Shalev-Shwartz and Zhang 2013; Johnson and Zhang 2013; Lin and Tong 2014) address this issue by designing estimators with low variance. Although these approaches have made an important progress in achieving the ideal *linear* convergence rate which is significant from the computational perspective, their analyses are restricted to the batch setting which requires to know the total number of data instances beforehand.

Online learning represents a family of effective and scalable learning algorithms for incrementally building a predictive model from a sequence of data samples (Rosenblatt 1958). Different from the conventional learning algorithms which usually require an expensive procedure to retrain entire dataset, when a new instance arrives (Chang and Lin 2011), the goal of online learning is to utilize this data instance to improve model without revisiting previously processed data. The seminal line of work in online learning, referred to as *linear online learning* (Rosenblatt 1958; Crammer et al. 2006), aims to learn a linear predictor in input space, which has a key limitation in representing data with nonlinear dependency, commonly seen in many real-world applications. This motivates the works of *kernel online learning* (Freund and Schapire 1999) that use a linear model in the feature space to capture non-linearity of input data. Although kernel online learning methods can capture non-linear nature of input data, a naive application stochastic gradient descent or incremental gradient descent to kernel-based methods for online learning encounters the *curse of kernelization*, that is, model size linearly grows with training size accumulated over time (Steinwart 2003; Wang et al. 2012).

To address the curse of kernelization, a remarkable approach is to use a budget (Dekel et al. 2005; Wang et al. 2012; Le et al. 2016a, b). Wang et al. (2012) conjoined the budgeted approach and stochastic gradient descent (SGD) (Shalev-Shwartz et al. 2007), wherein model was updated using an SGD-based style and a budget maintenance procedure (e.g., removal, projection, or merging) was employed to maintain the model size. Although the projection and merging were shown to be effective (Wang et al. 2012), their associated computational costs render them impractical for large-scale datasets. Another notable workaround to address the curse of kernelization is to employ random features (Rahimi and Recht 2007) in order to approximate a given kernel function (Lu et al. 2015; Le et al. 2016). Lu et al. (2015) transformed data from input space to random-feature space, and then performed SGD in this approximate space. However, to achieve a good kernel approximation in this approach, excessive number of random features might be required, hence possibly leading to serious computational issue. To overcome the issues emerged in Wang

et al. (2012), Lu et al. (2015), Le et al. (2016 proposed to distribute model in a dual space including original and random feature spaces, which allows information carried in ignored vectors to be more accurately preserved. Though these aforementioned methods can avoid the curse of kernelization, none of them has achieved the ideal *linear* convergence rate.

Incremental methods (Schmidt et al. 2013; Shalev-Shwartz and Zhang 2013; Johnson and Zhang 2013; Lin and Tong 2014) are *not applicable* to online learning because these methods require an entire training set beforehand during their training process and need to revisit past data instances when updating current models. In this paper, we make it possible by leveraging an *instant memory* that can store a snapshot of the most recent m data instances with the update style of Stochastic Variance-reduced Gradient Descent (SVRG) (Johnson and Zhang 2013; Ming et al. 2014) to propose *Dual space Stochastic Variance-reduced Gradient Descent* (DualSVRG). Particularly, for our DualSVRG, the new model is updated based on the current memory and an SVRG update style to reduce the variance when estimating gradient. Moreover, to address the curse of kernelization and scale up training time, we propose *transformer oracles* (see Sect. 3.6) to either move an incoming instance to a Fourier random feature space (Rahimi and Recht 2007) or keep it in the original kernel space, hence allowing the models to be distributed in a dual space. Furthermore, we establish rigorous theory to prove the ϵ-approximate *linear* convergence rate of our proposed DualSVRG. Overall, the key contributions in our paper can be summarized as follows:

- We propose a novel kernel online learning method and theoretically prove its approximate linear convergence rate. In terms of modeling, our proposed approach leverages SVRG (Johnson and Zhang 2013; Lin and Tong 2014) and DualSGD (Le et al. 2016) with adaptation to target the problem of kernel online learning. However, the adaptation undertaken in our proposed approach is significant as this requires to tackle three crucial challenges: (1) how to approximate efficiently the full gradients required in SVRG in the context of kernel online learning, (2) how to distribute the model into the dual space, and (3) how to develop theory for a convergence rate. We address the first challenge by using the concept of memory, which, to our knowledge, is new and novel in the context of kernel online learning, whilst the second challenge is addressed via three transformer oracles. The last challenge is addressed thoughtfully by developing theory to indicate that our DualSVRG achieves a faster convergence rate than DualSGD, which is also empirically verified by our comprehensive experiments.
- We conduct comprehensive experiments to compare our proposed methods with state-of-the-art baselines. The experimental results show that our proposed DualSVRG achieves superior predictive performance due to its faster convergence and ability to rapidly find a solution up to any level of precision, while spending comparable training time compared with state-of-the-art baselines.

2 Related background

2.1 Fourier random feature

Let $\boldsymbol{x} \in \mathbb{R}^d$ denote the d-dimensional vector in data domain $\mathcal{X}$ and $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel function. According to Mercer theorem, if $K(\cdot, \cdot)$ is a positive semi-definite (p.s.d) kernel, there exists a transformation $\boldsymbol{\Phi}(\cdot)$ that maps from $\mathcal{X}$ to a feature space $\mathcal{H}$ such that

$K(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{\Phi}(\boldsymbol{x})^\top \boldsymbol{\Phi}(\boldsymbol{x}')$ for all $\boldsymbol{x},\boldsymbol{x}' \in \mathcal{X}$. The kernel function $K(\boldsymbol{x},\boldsymbol{x}')$ can be regarded as the similarity of $\boldsymbol{x}$ and $\boldsymbol{x}'$ in the feature space. However, for the most popular Gaussian kernel $K(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}')^\top \Sigma(\boldsymbol{x}-\boldsymbol{x}')\right)$, the dimension of the feaure space $\mathcal{H}$ is infinite, hence obliging models in this space to be represented as $\mathbf{w} = \sum_{i=1}^{t} \alpha_i \boldsymbol{\Phi}(\boldsymbol{x}_i)$ where t is the training size in batch setting or the number of data instances processed so far. The dot product between $\mathbf{w}$ and any $\boldsymbol{\Phi}(\boldsymbol{x})$ which is crucial in developing kernel methods is represented as:

$$\mathbf{w}^\top \boldsymbol{\Phi}(\boldsymbol{x}) = \sum_{i=1}^{t} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}),$$

which is highly expensive when t is big. This issue is known as *the curse of kernelization*.

To construct an explicit representation of $\boldsymbol{\Phi}(\boldsymbol{x})$, the key idea is to approximate the original kernel $K(\boldsymbol{x},\boldsymbol{x}')$ using a kernel induced by a random finite-dimensional feature map. The mathematical tool behind this approximation is the Bochner's theorem (2003) which states that every shift-invariant and p.s.d kernel $K(\boldsymbol{x},\boldsymbol{x}')$ can be represented as an inverse Fourier transform of a proper distribution $p(\boldsymbol{\omega})$ as below:

$$K(\boldsymbol{x},\boldsymbol{x}') = k(\boldsymbol{u}) = \int p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^\top \boldsymbol{u}} d\boldsymbol{\omega}, \tag{1}$$

where $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x}'$ and i represents the imaginary unit (i.e., $i^2 = -1$). In addition, the corresponding proper distribution $p(\boldsymbol{\omega})$ can be recovered through Fourier transform as follows:

$$p(\boldsymbol{\omega}) = \left(\frac{1}{2\pi}\right)^d \int k(\boldsymbol{u}) e^{-i\boldsymbol{u}^\top \boldsymbol{\omega}} d\boldsymbol{u}. \tag{2}$$

From Eq. (1), we can use Monte Carlo estimation to approximate the kernel $K(\boldsymbol{x},\boldsymbol{x}')$ as:

$$K(\boldsymbol{x},\boldsymbol{x}') \approx \frac{1}{L} \sum_{i=1}^{L} \left[\cos\left(\boldsymbol{\omega}_i^\top (\boldsymbol{x}-\boldsymbol{x}')\right)\right]. \tag{3}$$

where we have sampled $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\boldsymbol{\omega})$ as formulated in Eq. (2). Denote $D = 2L$. Eq. (3) sheds light on the construction of a D-dimensional Fourier random map $\tilde{\boldsymbol{\Phi}} : \mathcal{X} \to \mathbb{R}^D$:

$$\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})^\top = \left[\cos\left(\boldsymbol{\omega}_i^\top \boldsymbol{x}\right)/\sqrt{L}, \sin\left(\boldsymbol{\omega}_i^\top \boldsymbol{x}\right)/\sqrt{L}\right]_{i=1}^{L}, \tag{4}$$

resulting in the induced kernel $\tilde{K}(\boldsymbol{x},\boldsymbol{x}') = \tilde{\boldsymbol{\Phi}}(\boldsymbol{x})^\top \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}')$ that can accurately and efficiently approximate the original kernel: $\tilde{K}(\boldsymbol{x},\boldsymbol{x}') \approx K(\boldsymbol{x},\boldsymbol{x}')$.

2.2 SGD update style for kernel online learning and curse of kernelization

In a kernel online learning system, at the time step t, the system receives data instance $(\boldsymbol{x}_t, y_t)$ and the current model is updated using an SGD-based formula as follows:

$$\begin{aligned} \mathbf{w}_t &= \mathbf{w}_{t-1} - \eta_t \nabla_{\mathbf{w}} l(\mathbf{w}_{t-1}; \boldsymbol{x}_t, y_t) \\ &= \mathbf{w}_{t-1} - \eta_t y_t \alpha_t \boldsymbol{\Phi}(\boldsymbol{x}_t), \end{aligned}$$

where $\eta_t > 0$ is a learning rate and α_t is a scalar which depends on the loss function $l(\mathbf{w}_{t-1}; \boldsymbol{x}_t, y_t)$. For example, if $l(\mathbf{w}_{t-1}; \boldsymbol{x}_t, y_t) = \max\{0, 1 - y_t \mathbf{w}_t^\top \boldsymbol{\Phi}(\boldsymbol{x}_t)\}$ is the Hinge loss then $\alpha_t = -\mathbb{I}_{y_t \mathbf{w}_t^\top \boldsymbol{\Phi}(\boldsymbol{x}_t) < 1}$ where $\mathbb{I}$ is the indicator function.

It follows that the model at the time step t has the form $\mathbf{w}_t = \sum_{i=1}^{t} \beta_i \boldsymbol{\Phi}(\boldsymbol{x}_i)$ and the model size is defined as $\|\boldsymbol{\beta}\|_0$. It follows that the model size is proportional to the number of data instances received so far which is accumulated over time. This issue is known as the curse of kernelization (Steinwart 2003; Wang et al. 2012) in the online learning context.

2.3 Dual space gradient descent for kernel online learning

DualSGD addressed the curse of kernelization by allowing the models to be stored in a dual space– a combination of the original feature space and the Fourier random feature space that approximates it. In particular, the model $\mathbf{w}_t$ at the time step t is distributed over two spaces (i.e., the original feature space and the Fourier random feature space) as $\mathbf{w}_t^d = \mathbf{w}_t^o \oplus \tilde{\mathbf{w}}_t$ where $\mathbf{w}_t^o = \sum_{i \in I} \beta_i \boldsymbol{\Phi}(\boldsymbol{x}_i)$ and $\tilde{\mathbf{w}}_t = \sum_{i \in J} \beta_i \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i) \in \mathbb{R}^D$ where $I \cap J = \emptyset$ and $I \cup J = \{1, 2, \ldots, t\}$. Since $\tilde{\mathbf{w}}_t \in \mathbb{R}^D$ can be stored directly in the Fourier random feature, the model size is defined as the cardinality $|I|$ of the index set I. In DualSGD (Le et al. 2016), the model size $|I|$ is kept at most B via the budget maintaining k-merging, that is, whenever the model size $|I|$ exceeds the budget size B, the k vectors in I with smallest coefficients are shifted to the Fourier random feature space and $\tilde{\mathbf{w}}_t$ is updated using the approximate versions of these vectors in the Fourier random feature. In particular, assume that the k vectors $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_k}$ are chosen to move to the Fourier random feature and the relevant models are updated as follows:

$$\mathbf{w}_{t+1}^o = \mathbf{w}_t^o - \sum_{j=1}^{k} \beta_{i_j} \boldsymbol{\Phi}\left(\boldsymbol{x}_{i_j}\right),$$

$$\tilde{\mathbf{w}}_{t+1} = \tilde{\mathbf{w}}_t + \sum_{j=1}^{k} \beta_{i_j} \tilde{\boldsymbol{\Phi}}\left(\boldsymbol{x}_{i_j}\right).$$

Because DualSGD employed the SGD update style wherein gradients using in update may approximate full gradients with high variances, the theoritical analysis in DualSGD only achieved the ε-approximate $\mathcal{O}\left(\frac{\log T}{T}\right)$ (see Definition 2). In this work, we used the variance reduction technique (Johnson and Zhang 2013; Lin and Tong 2014) to allow gradients in use to be approximated with much lower variances, hence achieving the ε-approximate linear convergent rate (see Definition 1). However, because Stochastic Variance Reduction used the entire training set to periodically compute full gradients involving in update formulas, this is not applicable to online learning. In the next section, we present variance reduction technique for batch setting and also give detailed discussions on the reason why this technique cannot be applied to online learning context.

2.4 Stochastic variance reduction for batch setting

Given a training set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, the common optimization problem of kernel methods with the ℓ_2-regularizer in the batch setting has the following form

$$\min_{\mathbf{w}} \left(\mathcal{J}(\mathbf{w}) \triangleq \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} l\big(\mathbf{w};\boldsymbol{x}_i, y_i\big) \right),$$

where $l(\mathbf{w};\boldsymbol{x}, y) = \ell\big(y\mathbf{w}^{\mathsf{T}}\boldsymbol{\Phi}(\boldsymbol{x})\big)$ with $\ell : \mathbb{R} \to \mathbb{R}$ to be a convex, L-Lipschitz and M-smooth loss function, and $\boldsymbol{\Phi}(\cdot)$ is a transformation from the input space to the feature space.

At the time step t, we uniformly sample i_t from $\{1, 2, \ldots, N\}$ and update $\mathbf{w}_{t-1}$ using the gradient $\nabla\mathcal{J}_t\big(\mathbf{w}_{t-1}\big)$ of the instantaneous objective function, which is defined as

$$\mathcal{J}(\mathbf{w}) \triangleq \frac{\lambda}{2}\|\mathbf{w}\|^2 + l\big(\mathbf{w};\boldsymbol{x}_{i_t}, y_{i_t}\big),$$

whose gradient $\nabla\mathcal{J}_t\big(\mathbf{w}_{t-1}\big)$ is an estimation of the full gradient $\nabla\mathcal{J}\big(\mathbf{w}_{t-1}\big)$. The variance of this estimation is crucial for the convergence rate of the SGD-based method. To reduce the variance of the full gradient estimation, the works of Johnson and Zhang (2013), Lin and Tong (2014) proposed using predictive variance reduction to update the current model as follows:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta\nabla\mathcal{J}_t\big(\mathbf{w}_{t-1}\big) + \eta\nabla\mathcal{J}_t(\boldsymbol{u}) - \eta\nabla\mathcal{J}(\boldsymbol{u}),$$

where $\eta > 0$ is the learning rate, the snapshot model $\boldsymbol{u}$ is updated periodically after every n iterations.

It is worth noting that since the evaluation of the full gradient $\nabla\mathcal{J}(\boldsymbol{u})$ involves all data instances in the training set, Stochastic Variance-reduced Gradient Descent (Johnson and Zhang 2013; Lin and Tong 2014) cannot be performed in the online learning context. To enable our proposed DualSVRG to work in the online context, we propose to approximate the full gradient $\nabla\mathcal{J}(\boldsymbol{u})$ using the most recent m data instances (e.g., $m = 100$), which are drawn from an existed but unknown joint data-label distribution over $\mathcal{X}\times\mathcal{Y}$. Specifically, the online system is equipped with a *snapshot memory* $\mathcal{M}$ that can store the most m recent data instances to make the computations feasible.

3 Dual space SVRG for kernel online learning

3.1 Optimization problem setting

In an online learning context (Kivinen et al. 2004), we aim to solve the following optimization problem:

$$\min_{\mathbf{w}}\Big(\mathcal{J}(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \mathbb{E}_{\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}}\big[l(\mathbf{w};\boldsymbol{x}, y)\big]\Big),$$

where $\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$ is a distribution over $\mathcal{X}\times\mathcal{Y}$, $l(\mathbf{w};\boldsymbol{x}, y) = \ell\big(y\mathbf{w}^{\mathsf{T}}\boldsymbol{\Phi}(\boldsymbol{x})\big)$ where $\ell : \mathbb{R} \to \mathbb{R}$ is a convex, L-Lipschitz and M-smooth loss function, and $\boldsymbol{\Phi}(\cdot)$ is a transformation from input space to feature space. We now introduce the definitions of the ε-approximate linear convergence rate and the ε-approximate $\mathcal{O}\Big(\frac{\log T}{T}\Big)$ convergence rate, which are necessary for our theoretical analysis.

Definition 1 An online learning algorithm is said to achieve the ε-approximate *linear* convergence rate if the gap between the objective values of the current and optimal models

decreases multiplicatively with the decay ratio $0 < \rho < 1$ with the to an ε-gap. Mathematically, it is stated as follows:

$$\mathbb{E}\big[\mathcal{J}(\mathbf{w}_t)\big] - \mathcal{J}(\mathbf{w}^*) \leq \rho^{t/a} b + \varepsilon,$$

where $a > 0, b \in \mathbb{R}$.

Definition 2 An online learning algorithm is said to achieve the ε-approximate $\mathcal{O}\Big(\frac{\log T}{T}\Big)$ rate if the gap between the objective values of the current and optimal models decreases with the rate $\mathcal{O}\Big(\frac{\log T}{T}\Big)$ to an ε-gap. Mathematically, it is stated as follows:

$$\mathbb{E}\big[\mathcal{J}(\mathbf{w}_t)\big] - \mathcal{J}(\mathbf{w}^*) \leq \frac{a(\log\, T + b)}{T} + \varepsilon,$$

where $a > 0$ and $b \geq 0$.

3.2 Models in the dual space

The dual space (Le et al. 2016) is a combination of the original feature space with the feature map $\boldsymbol{\Phi}(\boldsymbol{x})$ and its corresponding random feature space with the Fourier random map $\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$. In the online learning setting, when a data instance $\boldsymbol{x}$ arrives to the system, it is stored either in the original feature space or the random feature space. As a result, if we assume at the time step t, the system has received data instances $\boldsymbol{x}_1, ..., \boldsymbol{x}_t$, this set is split to two disjoint sets: $\{\boldsymbol{x}_i\}_{i\in I}$ and $\{\boldsymbol{x}_i\}_{i\in \tilde{I}}$ (i.e., $I \cup \tilde{I} = \{1, ..., t\}$), each of which is either stored in the original or random space. Specifically, the former $\{\boldsymbol{x}_i\}_{i\in I}$ is stored in the original feature space, hence resulting in the model $\mathbf{w}_t^o = \sum_{i\in I} \alpha_i \boldsymbol{\Phi}(\boldsymbol{x}_i)$ (o means *original*) in this space, whilst the latter $\{\boldsymbol{x}_i\}_{i\in \tilde{I}}$ is stored in the random feature space, hence resulting in the model $\tilde{\mathbf{w}}_t = \sum_{i\in \tilde{I}} \alpha_i \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i) \in \mathbb{R}^D$ (D is the dimension of the random feature space) in this space. As a result, the dual-model $\mathbf{w}_t^d := \mathbf{w}_t^o \oplus \tilde{\mathbf{w}}_t$ (i.e., d means *dual*) can be regarded as an approximation of the exact model $\mathbf{w}_t = \sum_{i=1}^t \alpha_i \boldsymbol{\Phi}(\boldsymbol{x}_i)$ for which all data instances are stored in the original feature space. Here we note that the operator $\oplus$ is used to imply the fact that our dual model is distributed over two spaces. Using this dual space strategy brings up some advantages: i) $\mathbf{w}_t^d$ can approximate the exact model $\mathbf{w}_t$ accurately, ii) $\tilde{\mathbf{w}}_t \in \mathbb{R}^D$ can be stored using a single vector, hence avoiding the curse of kernelization, and iii) the kernel operation which can be performed economically and conveniently as

$$\big\langle \mathbf{w}_t^d, \boldsymbol{x} \big\rangle := \sum_{i\in I} \alpha_i K(\boldsymbol{x}_i, x) + \tilde{\mathbf{w}}_t^{\top} \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}),$$

can approximate accurately and efficiently the kernel operation $\mathbf{w}_t^{\top} \boldsymbol{\Phi}(\boldsymbol{x})$, which appears all the time in kernel methods, as $\tilde{\mathbf{w}}_t^{\top} \tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$ can be computed directly as dot product of two vectors in $\mathbb{R}^D$.

3.3 The proposed method

In an online learning context, we assume that at the time step t when the data instance/label $(\boldsymbol{x}_t, y_t) \sim \mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$ arrives in our learning system, there is a *transformer oracle*, which makes a decision about if this instance is stored in the original feature space or in the random

feature space. We depict this decision using a binary random variable Z_t, where $Z_t = 1$ implies that this instance lies in the random feature space and vice versa. The current dual model $\mathbf{w}_{t-1}^d$ in the dual space is hence as follows:

$$\begin{aligned}
\mathbf{w}_{t-1}^d &= \mathbf{w}_{t-1}^o \oplus \tilde{\mathbf{w}}_{t-1}, \\
\mathbf{w}_{t-1}^o &= \sum_{i \in I_{t-1}} \alpha_i \boldsymbol{\Phi}(\boldsymbol{x}_i) = \sum_{i=1}^{t-1} (1 - Z_i) \alpha_i \boldsymbol{\Phi}(\boldsymbol{x}_i), \\
\tilde{\mathbf{w}}_{t-1} &= \sum_{i \in \tilde{I}_{t-1}} \alpha_i \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i) = \sum_{i=1}^{t-1} Z_i \alpha_i \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i) \in \mathbb{R}^D,
\end{aligned}$$

where $I_{t-1} = \{i : Z_i = 0 \text{ for } 1 \le i \le t-1\}$ and $\tilde{I}_{t-1} = \{i : Z_i = 1 \text{ for } 1 \le i \le t-1\}$.

We now present how to update those models to incorporate $(\boldsymbol{x}_t, y_t)$. Specifically, in our proposed model, we always maintain a *memory* $\mathcal{M}$ of the most recent m data instances/labels $\mathcal{M} = [(\boldsymbol{x}_i, y_i)]_{i=t-m+1}^t$ and a *dual checkpoint model* $\boldsymbol{u}^d := \frac{1}{n} \sum_{i=t-n+1}^t \mathbf{w}_i^d$, which is updated after every n iterations. The new models are updated based on the current memory $\mathcal{M}$ and checkpoint model $\boldsymbol{u}^d$ as follows:

$$\begin{aligned}
\boldsymbol{v}_t =& \nabla \ell\big(y_t \langle \mathbf{w}_{t-1}^d, \boldsymbol{x}_t \rangle\big) y_t \boldsymbol{\Phi}(\boldsymbol{x}_t) - \nabla \ell\big(y_t \langle \boldsymbol{u}^d, \boldsymbol{x}_t \rangle\big) y_t \boldsymbol{\Phi}(\boldsymbol{x}_t) \\
&+ \frac{1}{m} \sum_{i=t-m+1}^{t} \nabla \ell\big(y_i \langle \boldsymbol{u}^d, \boldsymbol{x}_i \rangle\big) y_i \boldsymbol{\Phi}(\boldsymbol{x}_i),
\end{aligned} \tag{5}$$

$$\boldsymbol{v}_t^d = \boldsymbol{v}_t^o \oplus \tilde{\boldsymbol{v}}_t, \tag{6}$$

where $\boldsymbol{v}_t^o$ and $\tilde{\boldsymbol{v}}_t$ lie in the original and Fourier random feature spaces respectively and are defined as:

$$\begin{aligned}
\boldsymbol{v}_t^o =& (1 - Z_t) \nabla \ell\big(y_t \langle \mathbf{w}_{t-1}^d, \boldsymbol{x}_t \rangle\big) y_t \boldsymbol{\Phi}(\boldsymbol{x}_t) - (1 - Z_t) \nabla \ell\big(y_t \langle \boldsymbol{u}^d, \boldsymbol{x}_t \rangle\big) y_t \boldsymbol{\Phi}(\boldsymbol{x}_t) \\
&+ \frac{1}{m} \sum_{i=t-m+1}^{t} \big[(1 - Z_i) \nabla \ell\big(y_i \langle \boldsymbol{u}^d, \boldsymbol{x}_i \rangle\big) y_i \boldsymbol{\Phi}(\boldsymbol{x}_i)\big], \\
\tilde{\boldsymbol{v}}_t =& Z_t \nabla \ell\big(y_t \langle \mathbf{w}_{t-1}^d, \boldsymbol{x}_t \rangle\big) y_t \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_t) - Z_t \nabla \ell\big(y_t \langle \boldsymbol{u}^d, \boldsymbol{x}_t \rangle\big) y_t \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_t) \\
&+ \frac{1}{m} \sum_{i=t-m+1}^{t} \big[Z_i \nabla \ell\big(y_i \langle \boldsymbol{u}^d, \boldsymbol{x}_i \rangle\big) y_i \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i)\big].
\end{aligned}$$

$$\mathbf{w}_t^d = \frac{\mathbf{w}_{t-1}^d - \eta \boldsymbol{v}_t^d}{\eta \lambda + 1}, \tag{7}$$

$$\mathbf{w}_t = \frac{\mathbf{w}_{t-1} - \eta \boldsymbol{v}_t}{\eta \lambda + 1}. \tag{8}$$

Here we remind that $l(\mathbf{w}; \boldsymbol{x}, y) = \ell\big(y \mathbf{w}^\top \boldsymbol{\Phi}(\boldsymbol{x})\big)$ where $\ell : \mathbb{R} \to \mathbb{R}$ is the loss function, implying $\nabla \ell(\cdot)$ is a scalar and $\eta > 0$ is the learning rate. We note that $\mathbf{w}_t$ and $\boldsymbol{v}_t$ in Eqs. (8, 5) lie in the original feature space and only are used later for our theoretical analysis.

The key steps of our proposed DualSVRG are presented in Fig. 1. In addition, the pseudocode of our proposed DualSVRG is detailed in Algorithm 1. When receiving a new data

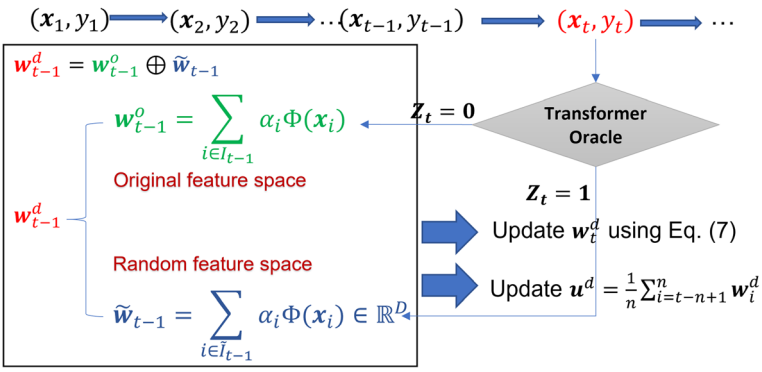

Fig. 1 The overall framework of our *DualSVRG approach*. When an incoming data instance (x_t, y_t) arrives, a transformer oracle is invoked to decide if we move this data instance to the random feature space (i.e., $Z_t = 1$) or the original feature space (i.e., $Z_t = 0$). We then update $\mathbf{w}^d_t$ and $\boldsymbol{u}^d$ for the next round. The *dual checkpoint model* $\boldsymbol{u}^d := \frac{1}{n}\sum_{i=t-n+1}^{t} \mathbf{w}^d_i$ is updated after every n iterations

instance (x_t, y_t), a *transformer oracle* (see Sect. 3.6) is invoked to decide if this instance stays in the original (i.e., $Z_t = 0$) or random feature space (i.e., $Z_t = 1$). Subsequently, we compute two dual vectors $\boldsymbol{v}^d_t$ and $\mathbf{w}^d_t$ using Eqs. (6, 7) respectively. After every n consecutive iterations in $(r-1)n+1, \ldots, rn$, termed as an *updating period*, we recompute two vectors $\boldsymbol{u}^d$ in the dual space and $\boldsymbol{u}$ (for analysis only) in the original feature space by averaging of n consecutive models $\mathbf{w}^d_t$(s) and $\mathbf{w}_t$(s) as in lines 8 and 9. In addition, in the end of the updating periods, we also update $\mathbf{w}^d_t$ and $\mathbf{w}_t$ (for analysis only) as in lines 10, 11.

3.4 The role of memory and comparison to full SVRG

If we apply the full SVRG strategy to the kernel online problem, we need to compute $\boldsymbol{v}^{\text{full}}_t$ as

$$\begin{aligned}\boldsymbol{v}^{\text{full}}_t =& \nabla \ell\big(y_t \mathbf{w}_{t-1}^{\mathsf{T}} \boldsymbol{\Phi}(\boldsymbol{x}_t)\big) y_t \boldsymbol{\Phi}(\boldsymbol{x}_t) - \nabla \ell\big(y_t \boldsymbol{u}^{\mathsf{T}} \boldsymbol{\Phi}(\boldsymbol{x}_t)\big) y_t \boldsymbol{\Phi}(\boldsymbol{x}_t) \\ &+ \frac{1}{t}\sum_{i=1}^{t} \nabla \ell\Big(y_i \big(\boldsymbol{u}^d\big)^{\mathsf{T}} \boldsymbol{\Phi}(\boldsymbol{x}_i)\Big) y_i \boldsymbol{\Phi}(\boldsymbol{x}_i),\end{aligned} \tag{9}$$

and then rely on $\boldsymbol{v}^{\text{full}}_t$ to compute $\mathbf{w}^{\text{full}}_t$ as in Eq. (8) but substituting $\boldsymbol{v}_t$ by $\boldsymbol{v}^{\text{full}}_t$. However, this involves all data instances/labels $\big[(x_i, y_i)\big]_1^t$, hence infeasible for the online learning for which we require instant and immediate update without any revisiting the entire dataset.

Our workaround is to use a memory $\mathcal{M} = \big[(x_i, y_i)\big]_{t-m+1}^{t}$ to store a snapshot of the most recent m data instances/labels. The computation in Eq. (6) becomes infeasible because it only uses a snapshot of the most recent m data instances/labels. Additionally, in our experiments, though we only set $m = 100$ (i.e., a tiny memory), our proposed methods achieve very good performance. We note that at very first rounds, when the system has not received sufficiently m data instances/labels, we use up all existing data instances for the memory $\mathcal{M}$ and the formula is modified accordingly.

The formulas to update $\boldsymbol{v}^d_t$ and $\mathbf{w}^d_t$ are quite complex due to the involvement of the dual space. However, they are derived in such a way that $\boldsymbol{v}^d_t$ in Eq. (6) is a replicate in the dual space of $\boldsymbol{v}_t$ in the original feature space, which in turn is an approximation of $\boldsymbol{v}^{\text{full}}_t$. By these

means, we can develop our theory in sequel to show that the gap between our current model and the optimal one is proportional to $\frac{1}{m}$, which decreases rapidly when using more memory.

Finally, similar to SVRG, we always maintain a dual checkpoint model $\boldsymbol{u}^d = \frac{1}{n}\sum_{i=t-n+1}^{t} \mathbf{w}_i^d$ which is used to compute $\boldsymbol{v}_t^d$. The dual checkpoint model $\boldsymbol{u}^d$ is updated after every n iterations. In our experiments, we set the updating period $n = 100$ and conduct the ablation study on the influence of the updating period n to the performance.

Algorithm 1 The pseudocode of the proposed method. We note that lines *9* and *11* are only used for theoretical analysis.

Input: λ, $K(\cdot,\cdot)$, m, n
Output: $\mathbf{w}_T^d$
1: $\mathbf{w}_0^d = \mathbf{0}$; $\boldsymbol{u}^d = \mathbf{0}$; $\boldsymbol{u} = \mathbf{0}$
2: **for** $t = 1$ **to** T **do**
3: Receive $(\boldsymbol{x}_t, y_t) \sim \mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$
4: Invoke a transformer oracle to find $Z_t \in \{0;1\}$
5: Compute $\boldsymbol{v}_t^d$ as in Eq. (6)
6: Compute $\mathbf{w}_t^d$ as in Eq. (7)
7: **if** t mod $n == 0$ **then**
8: $\boldsymbol{u}^d = \frac{1}{n}\sum_{i=t-n+1}^{t} \mathbf{w}_i^d$
9: $\boldsymbol{u} = \frac{1}{n}\sum_{i=t-n+1}^{t} \mathbf{w}_i$ *//analysis*
10: $\mathbf{w}_t^d = \boldsymbol{u}^d$
11: $\mathbf{w}_t = \boldsymbol{u}$ *//analysis*
12: **end if**
13: **end for**

3.5 Convergence analysis

In what follows we present the convergence analysis for our proposed method. This rigorous analysis shows that our proposed method enjoys the ε-approximate *linear* convergence rate, hence converging rapidly to its global minima. Without loss of generality, we assume that the kernel function $K(\cdot,\cdot)$ is symmetric, shift-invariant, positive semi-definite, and $\|\boldsymbol{\Phi}(\boldsymbol{x})\| = K(\boldsymbol{x},\boldsymbol{x})^{1/2} = 1$ for every $\boldsymbol{x}$. We further assume that the convex loss function $\ell(\cdot)$ is L-Lipschitz and M- strongly smooth. Finally, we also denote $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\mathcal{J}(\mathbf{w})$ and $\boldsymbol{u}_r$ to be the vector $\boldsymbol{u}$ at the r-th updating period (i.e., $\boldsymbol{u}_r = \mathbf{w}_{rn}$). All proofs of convergence analysis can be found in the supplementary material.[1].

Our aim is to prove that DualSVRG achieves the ε-approximate linear convergence rate and can rapidly converge to the true optimal solution $\mathbf{w}^*$. The pathway of our proof is as follows:

- We define $f_t(\boldsymbol{x}) = \mathbf{w}_t^\top \boldsymbol{\Phi}(\boldsymbol{x})$, $f_t^d(\boldsymbol{x}) = \left\langle \mathbf{w}_t^d, \boldsymbol{x} \right\rangle$, and in Theorem 1, we prove that given $\varepsilon > 0$ with a high probability, we have $\left|f_t(\boldsymbol{x}) - f_t^d(\boldsymbol{x})\right| < \varepsilon$ for every t and $\boldsymbol{x} \in \mathcal{X}$ or equivalently $\left\|f_t - f_t^d\right\| = \sup_{x\in\mathcal{X}}\left|f_t(\boldsymbol{x}) - f_t^d(\boldsymbol{x})\right| < \varepsilon$ in the function space.

[1] https://app.box.com/s/t5gwwu4qcrl0knq99lika74yf2jx12mr

- Theorem 2 and Corollary 1 show the $\varepsilon-$approximate linear convergence rate (cf. Definition 1) of our DualSVRG where the gap, which is $\mathcal{O}\left(\frac{1}{m}\right)$, reduces rapidly to 0. This also implies that $\mathbf{w}_{rn} = \boldsymbol{u}_r$ converges rapidly to $\mathbf{w}^*$ when r approaches $+\infty$.

In particular, Theorem 1 shows that the decision function $f_t^d(\cdot)$ induced by $\mathbf{w}_t^d$ is a tight approximation of the decision function $f_t(\cdot)$ induced by $\mathbf{w}_t$ in function space, followed by Theorem 2, which respectively point out that the proposed DualSVRG achieves the ε -approximate *linear* convergence rate. It is worth noting that in Theorem 2 , the gap $\frac{2(6L^2+12M)\eta}{m(1-\rho)}$ can plunge rapidly when increasing m. We empirically found that $m = 100$ works satisfactorily for real datasets.

Theorem 1 *Given $\varepsilon > 0$, with a probability at least $1-\delta = 1-2^8\left(\frac{3\sigma L d_{\mathcal{X}}}{\lambda\varepsilon}\right)^2 \exp\left(-\frac{D\lambda^2\varepsilon^2}{36(d+2)L^2}\right)$, where $d_{\mathcal{X}}$ is the diameter of the data domain $\mathcal{X}$, L is a constant, σ is defined as $\mathbb{E}\left[\boldsymbol{\omega}^\top\boldsymbol{\omega}\right]$, and D and d are the dimensions of the random feature space and the data space, we have the following inequalities*

(1) $\left|f_t(\boldsymbol{x}) - f_t^d(\boldsymbol{x})\right| < \varepsilon$ *for every t and $\boldsymbol{x} \in \mathcal{X}$.*

(2) *Assume that $\mathbf{w}_t$ has the representation $\mathbf{w}_t = \sum_{i=1}^t \alpha_i \boldsymbol{\Phi}(\boldsymbol{x}_i)$, we then have for every $\boldsymbol{x}$*

$$\mathbb{E}\left[\left|\mathbf{w}_t^\top\boldsymbol{\Phi}(\boldsymbol{x}) - \left\langle \mathbf{w}_t^d, \boldsymbol{x}\right\rangle\right|\right] < \frac{\lambda\varepsilon}{3L}\sum_{i=1}^t \mathbb{P}\left(Z_i = 1\right)^{1/2}\mathbb{E}\left[\alpha_i^2\right]^{1/2}.$$

(3) $\left|\boldsymbol{u}^\top\boldsymbol{\Phi}(\boldsymbol{x}) - \left\langle \boldsymbol{u}^d, \boldsymbol{x}\right\rangle\right| < \varepsilon$ *for every t and $\boldsymbol{x} \in \mathcal{X}$.*

Theorem 2 *Assume that $m \geq 2$, $\eta < \frac{\beta}{(13+12\beta)M}$, $n \geq \max\left\{\frac{1}{\eta\lambda(1-\beta)(1-12M\eta)}, 13\right\}$ for some $0 < \beta < 1$, and $\varepsilon < \frac{1}{m}$, with a probability at least $1-\delta$, we then have:*

$$\mathbb{E}\left[\mathcal{J}\left(\mathbf{w}_{rn}\right)\right] - \mathcal{J}(\mathbf{w}^*) < \rho^r\left[\mathcal{J}(\mathbf{0}) - \mathcal{J}(\mathbf{w}^*) - \frac{2\left(6L^2 + 12Mm^2\varepsilon^2\right)\eta}{m(1-\rho)}\right] + \frac{2\left(6L^2+12M\right)\eta}{m(1-\rho)},$$

where $\rho = \frac{13M\eta}{(\eta\lambda+1)(1-12\eta M)} + \frac{1}{n\lambda\eta(1-12\eta M)} < 1$.

Here we note that from $\eta < \frac{\beta}{(13+12\beta)M}$, $n \geq \frac{1}{\eta\lambda(1-\beta)(1-12M\eta)}$, we arrive

$$\frac{13M\eta}{(1+\eta\lambda)(1-12M\eta)} < \frac{13M\eta}{1-12M\eta} < \beta,$$

$$\frac{1}{n\lambda\eta(1-12M\eta)} < 1-\beta,$$

hence $\rho = \frac{13M\eta}{(\eta\lambda+1)(1-12\eta M)} + \frac{1}{n\lambda\eta(1-12\eta M)} < \beta + 1 - \beta = 1$.

Corollary 1 *Assume that*

$$m \geq 2, \eta < \min\left\{\frac{\beta}{(13+12\beta)M}, \frac{\varepsilon m(1-\rho)}{2\left(6L^2+12M\right)}\right\},$$
$$n \geq \max\left\{\frac{1}{\eta\lambda(1-\beta)(1-12M\eta)}, 13\right\},$$

for some $0 < \beta < 1$, *and* $\varepsilon < \frac{1}{m}$, *with a probability at least* $1-\delta$, *we then have*:

$$\mathbb{E}\left[\mathcal{J}\left(\mathbf{w}_{rn}\right)\right] - \mathcal{J}(\mathbf{w}^*) < \rho^r\left[\mathcal{J}(\mathbf{0}) - \mathcal{J}(\mathbf{w}^*) - \frac{2\left(6L^2+12Mm^2\varepsilon^2\right)\eta}{m(1-\rho)}\right] + \varepsilon,$$

where $\rho = \frac{13M\eta}{(\eta\lambda+1)(1-12\eta M)} + \frac{1}{n\lambda\eta(1-12\eta M)} < 1$.

3.6 The transformer oracles

In what follows we present three transformer oracles which decide whether an incoming data instance $\left(\boldsymbol{x}_t, y_t\right)$ stays in the original (i.e., $Z_t = 0$) or random (i.e., $Z_t = 1$) feature space. We define $I \subset \{1, 2, ..., t\}$ as the set of indices i whose element $\boldsymbol{x}_i$ is stored in the original feature space and $\tilde{I} = \{1, 2, ..., t\} \backslash I$ as the set of indices i whose element $\boldsymbol{x}_i$ is stored in the random feature space. We note that the model size of our DualSVRG is defined as the cardinality of I denoted by $|I|$.

3.6.1 Budget oracle

We employ a predefined budget B. When an incoming data instance $\left(\boldsymbol{x}_t, y_t\right)$ arrives, it is first added to the original feature space and then if the model size exceeds the budget B, this data instance will be moved to the random feature space. The pseudocode of this oracle is presented in Algorithm 2.

Algorithm 2 The budget oracle.

Input: $I, \tilde{I}, (\boldsymbol{x}_t, y_t), B$
Output: $Z_t \in \{0; 1\}$
$Z_t = 0$; $I = I \cup \{t\}$
if $|I| > B$ **then**
  $Z_t = 1$; $\tilde{I} = \tilde{I} \cup \{t\}$
  $I = I \backslash \{t\}$
end if

3.6.2 Always-move oracle

It is the simplest oracle, where we always move incoming instance $\left(\boldsymbol{x}_t, y_t\right)$ to the random feature space (i.e., always set $Z_t = 1$). The pseudocode of the always-move oracle is presented in Algorithm 3. The model size of our DualSVRG using this transformer oracle is 0.

Algorithm 3 The always-move oracle.

Input: $I, \tilde{I}, (\boldsymbol{x}_t, y_t)$
Output: $Z_t \in \{0; 1\}$
1: $Z_t = 1$; $\tilde{I} = \tilde{I} \cup \{t\}$

Algorithm 4 The coverage oracle.

Input: $I, \tilde{I}, (\boldsymbol{x}_t, y_t), \theta$
Output: $Z_t \in \{0; 1\}$
1: $\mathrm{d}(\boldsymbol{x}_t, \mathcal{C}) = \min_{\boldsymbol{x} \in \mathcal{C}} \|\boldsymbol{x}_t - \boldsymbol{x}\|$
2: **if** $\mathrm{d}(\boldsymbol{x}_t, \mathcal{C}) \leq \theta$ **then**
3: $Z_t = 1$
4: $\tilde{I} = \tilde{I} \cup \{t\}$
5: **else**
6: $Z_t = 0$
7: $I = I \cup \{t\}$
8: $\mathcal{C} = \mathcal{C} \cup \{\boldsymbol{x}_t\}$
9: **end if**

3.6.3 Coverage oracle

We maintain a core set $\mathcal{C} \subset \{\boldsymbol{x}_1, ..., \boldsymbol{x}_{t-1}\}$. When an incoming data instance $(\boldsymbol{x}_t, y_t)$ arrives, we compute the distance from $\boldsymbol{x}_t$ to $\mathcal{C}$, which is defined as $\mathrm{d}(\boldsymbol{x}_t, \mathcal{C}) = \min_{\boldsymbol{x} \in \mathcal{C}} \|\boldsymbol{x}_t - \boldsymbol{x}\|$. If this distance is less than a predefined threshold θ, we keep $(\boldsymbol{x}_t, y_t)$ in the original feature space and add $\boldsymbol{x}_t$ to the core set. Otherwise, we move $(\boldsymbol{x}_t, y_t)$ to the random feature space. The pseudocode of the coverage oracle is presented in Algorithm 4. The model size of our DualSVRG with this transformer oracle is bounded by $N(\mathcal{X}, \theta/2)$, which is the covering number of the compact set $\mathcal{X}$ w.r.t the radius $\theta/2$ (Shalev-Shwartz and Ben-David 2014). Assuming that the data domain $\mathcal{X}$ is a compact set, the model size of the proposed DualSVRG with the coverage oracle cannot exceed $N(\mathcal{X}, \theta/2)$, which is the covering number of the compact set $\mathcal{X}$ w.r.t the radius $\theta/2$. When increasing the budget size B in the budget oracle and the approximation threshold θ in the coverage oracle, we also increase the probability of performing the budget maintenance (i.e., $\mathbb{P}(Z_t = 1)$). Referring to Theorem 1 ii, this also rises the gap between the approximation and true decision functions, hence might reduce the predictive performance. In addition, by making some conditions to simplify the data distribution $\mathbb{P}_{\mathcal{X}}$, we can further bound $\mathbb{P}(Z_t = 1)$ and gain theoretical results involving B, θ. We leave this theoretical development to future work.

4 Experiments

In this section, we conduct comprehensive experiments to quantitatively evaluate the capacity and scalability of our proposed DualSVRG(s) on classification task under two different settings:

- *Batch classification*[2] the regular binary and multi-class classification tasks that follow a standard setup, wherein each dataset is partitioned into training set and testing set. The

[2] This setting is also known as offline classification.

Table 1 Classification performance of ours and the baselines in the batch mode

Dataset	w8a		Cod-rna		Covtype		Airlines	
Algorithm	Error	Time	Error	Time	Error	Time	Error	Time
LIBSVM	**0.94**	51	**3.61**	115	–	–	–	–
LLSVM	1.36	92	5.84	20	–	–	–	–
BSGD-M	1.83 ± 0.07	265	**4.33 ± 0.21**	91	27.74 ± 0.16	2413	–	–
BSGD-R	2.90 ± 0.04	253	33.17 ± 0.11	19	38.91 ± 1.69	419	19.73 ± 0.06	4741
FOGD	2.08 ± 0.38	32	7.35 ± 4.20	8	40.66 ± 5.85	70	19.63 ± 0.21	1086
NOGD	1.94 ± 0.18	375	8.17 ± 3.35	10	31.80 ± 2.96	679	25.17 ± 0.20	3112
DualSGD-Hinge	3.44 ± 0.17	34	29.15 ± 0.02	19	30.54 ± 0.54	375	23.13 ± 0.00	1696
DualSVRG-H-B	1.36 ± 0.13	150	6.30 ± 1.17	87	26.73 ± 1.55	2144	19.28 ± 0.05	3583
DualSVRG-H-C	1.35 ± 0.09	127	6.61 ± 0.85	83	26.50 ± 1.61	1724	**19.25 ± 0.02**	4019
DualSVRG-H-AM	**1.29 ± 0.01**	109	5.98 ± 1.53	74	**26.15 ± 0.44**	1737	19.28 ± 0.04	4517
DualSVRG-L-B	2.24 ± 1.66	158	6.20 ± 0.79	86	27.61 ± 0.64	2127	19.28 ± 0.06	3447
DualSVRG-L-C	2.27 ± 1.75	127	7.13 ± 0.91	79	**26.14 ± 1.31**	1788	**19.26 ± 0.04**	3594
DualSVRG-L-AM	1.71 ± 0.47	112	5.77 ± 0.27	104	26.59 ± 1.64	1855	19.28 ± 0.07	4447

The error (%) and training time (second). The best and runner-up performance are in bold and italic-bold respectively

models are trained on the training part, and tested on the testing part; the testing error rate and training time are reported.

- *Online classification* the binary and multi-class classification tasks that follow a purely online learning setup, wherein there is no division of training and testing sets as in batch setting. The algorithms sequentially receive and process a single data instance turn-by-turn. When an individual data instance arrives, the models perform prediction to compute the mistake rate first, then use this data instance to update the models; the accumulated mistake rate and execution time are reported. Specifically, the mistake rate can be viewed as the online predictive performance for which we require a current model to predict an incoming data instance first before using this incoming data instance to update the current model. As a result, a lower mistake rate means an online learning method can adapt better and generalize more efficiently to future data instances.

We use 4 datasets which are *w8a*, *cod-rna*, *covtype*, and *airlines*. The datasets where purposely are selected with various sizes in order to clearly expose the differences among scalable capabilities of the models (*w8a*: *64,700*; *cod-rna*: *331,152*; *covtype*: *581,012* and *airlines*: *5,929,413*). These datasets can be downloaded from LIBSVM and UCI websites, except the airlines which was obtained from American Statistical Association (ASA3). For the airlines dataset, our aim is to predict whether a flight will be delayed or not under binary classification setting. A flight is considered delayed if its delay time is above 15 minutes, and non-delayed otherwise. Following the procedure in (Hensman et al. 2013), we extract 8 features for flights in the year of 2008, and then normalize them into the range [0,1].

In batch classification experiments, we follow the original divisions of training and testing sets in LIBSVM and UCI sites wherever available. For *covtype* and *airlines* datasets, we split the data into 90% for training and 10% for testing. In online classification task, we

Table 2 Classification performance of our proposed methods and baselines in the online mode

Dataset	w8a		cod-rna		covtype		Airlines	
Algorithm	Mistake rate	Time	Mistake rate	Time	Mistake rate	Time	Mistake rate	Time
Perceptron	3.51 ± 0.03	691	9.79 ± 0.04	1394	–	–	–	–
OGD	2.54 ± 0.03	1290	7.81 ± 0.03	2804	–	–	–	–
RBP	4.02 ± 0.07	545	26.02 ± 0.39	86	–	–	–	–
Forgetron	3.96 ± 0.10	558	28.56 ± 2.22	103	–	–	–	–
Projectron	4.76 ± 1.13	572	11.16 ± 3.61	97	–	–	–	–
Projectron++	3.08 ± 0.63	1322	17.97 ± 15.60	1780	–	–	–	–
BOGD	3.16 ± 0.08	589	38.13 ± 0.11	105	–	–	–	–
FOGD	3.52 ± 0.05	26	7.15 ± 0.03	53	40.45 ± 0.05	223	20.98 ± 0.01	1271
NOGD	2.55 ± 0.05	585	7.83 ± 0.06	105	34.72 ± 0.07	838	25.56 ± 0.01	3554
DualSGD-Hinge	2.99 ± 0.01	32	**4.92 ± 0.25**	98	49.64 ± 0.50	222	19.28 ± 0.00	1113
DualSVRG-H-B	2.99 ± 0.00	93	7.72 ± 0.14	274	23.08 ± 0.35	966	19.00 ± 0.09	6102
DualSVRG-H-C	**2.50 ± 0.01**	82	**4.95 ± 0.01**	248	***22.94 ± 0.31***	654	18.96 ± 0.01	4631
DualSVRG-H-AM	**2.43 ± 0.02**	69	7.64 ± 0.09	225	**22.92 ± 0.39**	751	18.91 ± 0.01	4333
DualSVRG-L-B	2.99 ± 0.00	97	5.02 ± 0.18	251	24.67 ± 0.01	820	**18.60 ± 0.08**	6336
DualSVRG-L-C	2.99 ± 0.00	77	5.06 ± 0.13	229	24.72 ± 0.20	666	18.65 ± 0.02	4605
DualSVRG-L-AM	2.99 ± 0.00	60	5.01 ± 0.18	253	24.73 ± 0.20	654	**18.58 ± 0.03**	4465

Here θ, B, $\hat{B}$, D, $\hat{D}$ are identical to the batch classification task. The mistake rate (%) and execution time (second)

The best and runner-up performance are in bold and italic-bold respectively

either use the entire datasets or concatenate training and testing parts into one. The online learning algorithms are then trained in a single pass through the data. In both batch and online settings, for each dataset, the models perform 10 runs on different random permutations of the training data samples. Their prediction results and time costs are then reported by taking the average with the standard deviation of the results over these runs. We create six variants of our DualSVRG by combining two loss functions with three stochastic oracles. In particular, in the abbreviations of variants in Tables 1 and 2, H and L stand for τ -smooth Hinge (Shalev-Shwartz and Zhang 2013) and Logistic losses respectively, whilst B, C, and AM stands for the budget, coverage, and always-move oracles respectively.

For comparison, we employ some baseline methods whose C++, Python implementations with Matlab interfaces are published as a part of LIBSVM, BudgetedSVM[3] LSOKL[4] toolboxes, and DualSGD GitHub[5]. *Our* DualSVRG *is implemented using Python and can be found here*[6]. Throughout the experiments, we utilize RBF kernel, i.e., $K(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-\gamma\|\boldsymbol{x}-\boldsymbol{x}'\|^2\right)$ for all algorithms including ours. All experiments are con-

[3] http://www.dabi.temple.edu/budgetedsvm/index.html.

[4] http://lsokl.stevenhoi.com/.

[5] https://github.com/tund/dsgd.

[6] https://anonymous.4open.science/r/8c28bb53-f223-4c39-8c83-04806e5ca8cd/.

ducted using a Windows machine with 3.46GHz Xeon processor and 96GB RAM. It is worth noting that the LLSVM does not support multi-class classification and we terminate all runs exceeding the limit of two hours, therefore some results are unavailable. Note that for a fair comparison, we implement DualSGD which bases on the memory to update the current model. However, we observe a comparable performance to the original DualSGD, whilst the training time is approximately multiplicative by m (e.g., $m = 100$) due to more gradient estimations. We hence do not report these results here.

4.1 Batch mode

4.1.1 Baselines for the batch mode

For classification in batch mode, we compare with the following state-of-the-art baselines to train kernel SVMs:

- *LIBSVM* one of the most widely-used and state-of-the-art implementations for batch kernel SVM solver (Chang and Lin 2011). We use the one-vs-all approach as the default setting for the multiclass tasks.
- *LLSVM* low-rank linearization SVM algorithm that approximates kernel SVM optimization by a linear SVM using low-rank decomposition of the kernel matrix (Zhanget al. 2012).
- *BSGD-M* budgeted SGD algorithm which extends the Pegasos algorithm (Shalev-Shwartz et al. 2007) by introducing a merging strategy for support vector budget maintenance (Wang et al. 2012).
- *BSGD-R* budgeted SGD algorithm which extends the Pegasos algorithm (Shalev-Shwartz et al. 2007) by introducing a removal strategy for support vector budget maintenance (Wang et al. 2012).
- *FOGD* Fourier online gradient descent algorithm that applies the random Fourier features for approximating kernel functions (Lu et al. 2015).
- *NOGD* Nystrom online gradient descent (NOGD) algorithm that applies the Nystrom method to approximate large kernel matrices (Lu et al. 2015).

4.1.2 Hyperparameters setting

There are a number of different hyperparameters for all methods. Each method requires a different set of hyperparameters, e.g., the regularization parameters (C in LIBSVM, λ in Pegasos and DualSVRG), the learning rates (η in FOGD and NOGD), the coverage diameter (θ in DualSVRG with the coverage oracle) and the RBF kernel width (γ in all methods). Thus, for a fair comparison, these hyperparameters are specified using cross-validation on training subset.

In particular, we further partition the training set into 80% for learning and 20% for validation. For large-scale databases, we use only 1% of training set, so that the searching can finish within an acceptable time budget. The hyperparameters are varied in certain ranges and selected for the best performance on the validation set. The ranges are given as follows: $C \in \left\{2^{-5}, 2^{-3}, ..., 2^{15}\right\}$, $\lambda \in \left\{\frac{2^{-4}}{N}, \frac{2^{-2}}{N}, ..., \frac{2^{16}}{N}\right\}$, $\eta \in \{16.0, 8.0, 4.0, 2.0, 0.2, 0.02, 0.002, 0.0002\}$, and $\gamma \in \left\{2^{-8}, 2^{-4}, 2^{-2}, 2^{0}, 2^{2}, 2^{4}, 2^{8}\right\}$,

Table 3 $\left[\theta \mid \hat{B} \mid B \mid \hat{D} \mid D \mid S\right]$ denotes diameter θ in coverage oracle, budget size $\hat{B}$ of DualSGD and ours with budget oracle, budget size B of other bugeted algorithm, number of random features $\hat{D}$ of DualSGD and ours, number of random features D of FOGD, and model size S

Dataset	w8	cod-rna	Covtype	Airlines
θ	13.0	1.0	3.0	1.0
$\hat{B}$	100	100	100	100
B	1000	400	400	1000
$\hat{D}$	200	200	200	200
D	4000	1600	1600	4000
S	131	436	400	388

where N is the number of data points. For the budget size B in NOGD and budgeted algorithms, and the feature dimension D in FOGD for each dataset, we use identical values to those used in Lu et al. (2015) (cf. Table 3). The budget size $\hat{B}$ and the feature dimension $\hat{D}$ of DualSGD and DualSVRG(s) are identical with those in Le et al. (2016) (cf. Table 3). The learning rate η of DualSVRG(s) is set to $\frac{1}{76M}$ (cf. Theorem 2 with $\beta = 0.5$). For DualSVRG(s), the parameters m and n are set to 100 and 100 respectively, the parameter τ in τ- smooth Hinge loss is set to 0.5.

4.1.3 Experimental results

The experimental results in the batch mode (cf. Table 1) show that with respect to predictive performance our proposed DualSVRG(s) are comparable with LIBSVM and almost outperform other baselines. Regarding the training time, our DualSVRG(s) are slower than FOGD, NOGD, and DualSGD-Hinge because of their additional computations in approximating full gradient and periodically updating $\boldsymbol{u}^d$. However, our DualSVRG(s) are still scalable with large-scale datasets. Overall, LIBSVM outperforms others for the batch setting, but LIBSVM is not scalable for large scale datasets in terms of training time and memory consumption because it solves a quadratic programming problem requiring to store a kernel matrix in the main memory. Although our proposed methods are only slightly superior or comparable with other baselines on the batch setting, for the online setting (i.e., the main setting in this paper as shown in Table 2), the snapshot memory really benefits in capturing correlation of data instances in it and allows us to estimate gradients more precisely. As a result, our proposed methods significantly outperform the baselines in the online setting.

4.2 Online mode

4.2.1 Baselines for the online mode

We employ the two widely-used algorithms – Perceptron and OGD for regular online kernel classification without budget maintenance and 8 state-of-the-art budget online kernel learning methods as follows:

- *Perceptron* the kernelized variant without budget of Perceptron algorithm (Freund and Schapire 1999).
- *OGD* the kernelized variant without budget of online gradient descent (Kivinen et al. 2004).
- *RBP* a budgeted Perceptron algorithm using random support vector removal strategy (Cavallanti et al. 2007).
- *Forgetron* a kernel-based Perceptron maintaining a fixed budget by discarding oldest support vectors (Dekel et al. 2005).
- *Projectron* a Projectron algorithm using the projection strategy (Orabona et al. 2009).
- *Projectron++* the aggressive version of Projectron algorithm (Orabona et al. 2009).
- *BOGD* a budgeted variant of online gradient descent algorithm using simple SV removal strategy (Zhao er al. 2012).
- *FOGD* and *NOGD* described in Sect. 4.1.

4.2.2 Hyperparameters setting

For each method learning on each dataset, we follow the same hyperparameter setting in the batch classification task.

4.2.3 Experimental results

The experimental results in the online mode (cf. Table 2) show that our proposed DualSVRG(s) achieve superior predictive performance over its rivals except for the *cod-rna* dataset. The training times of our DualSVRG(s) are slightly higher than those of FOGD, NOGD, and DualSGD-Hinge. However, it is arguably not a matter for the online learning problem because the time amount which our methods take to process each data instance (i.e., reported time divides by number of data instances processed, for instance, $\approx \frac{6.000}{5{,}929{,}413} \approx 10^{-4}$ *seconds/data instance* for the airlines dataset)) is negligible. Comparing among the variants, DualSVRG-H-C with the hinge loss and coverage oracle works overall better than others and is a good choice in practice.

Another observation from Table 2 is that our proposed methods show more significant improvements with large scale datasets (e.g., the covtype and airlines datasets). This can be partly explained as the snapshot memory can capture better correlation among data instances stored in it, which benefits for large scale datasets because this helps to reduce *catastrophic forgetting* more likely and seriously happening for long-term data streams.

4.3 Ablation study

4.3.1 Convergence rate

We now turn to empirically prove that our DualSVRG converges more rapidly than DualSGD, hence yielding lower objective value– the average of losses at data points processed so far. We choose to investigate this convergence behavior on the *w8a* and *cod-rna* datasets by training and computing the objective value for the same parameter set $\left(\lambda, \gamma, \hat{B}, \hat{D}\right)$. As can be observed from Fig. 2, the same pattern appears for two experimental datasets, that is, the plot of DualSGD-Hinge highly fluctuates and ends with a high value,

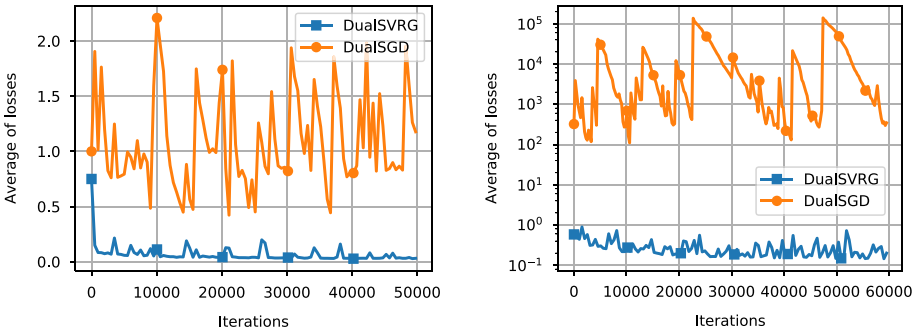

Fig. 2 The convergence rates of DualSVRG-H-B and DualSGD-Hinge on the *w8a* (left) and *cod-rna* (right) datasets

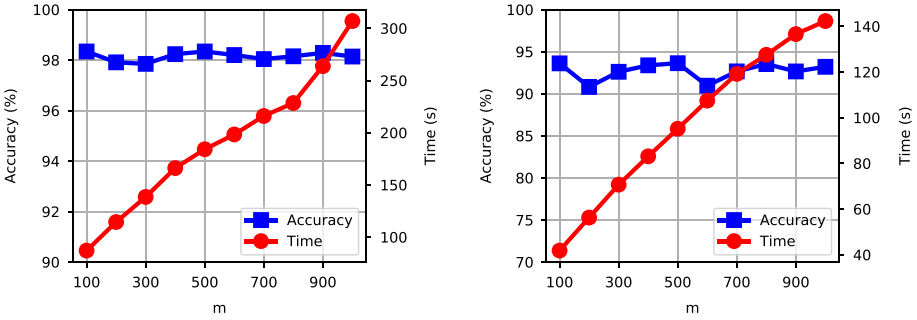

Fig. 3 The variation of accuracy and training time when varying m on *w8a* (left) and *cod-rna* (right) datasets

whilst ours rapidly and stably converges to the optimal value. This empirically confirms the better convergence rate of our DualSVRG in comparison with DualSGD.

4.3.2 Model behaviors when varying *m* and *n*

In this section, we inspect the model behaviors when varying m and n. To this end, we choose to do experiments on the *w8a* and *cod-rna* datasets. In the first experiment, we vary m to observe the variation of both accuracy and training time (cf. Fig. 3). As can be seen from Fig. 3, the same pattern repeats for both *w8a* and *cod-rna* datasets, that is, the accuracy slightly fluctuates, whilst the training time almost linearly grows. It can be reasoned from our developed theory as follows. Because of the $\mathcal{O}\left(\frac{1}{m}\right)$ gap in Theorem 2 which rapidly decreases to 0, the value of m does not significantly influence in the predictive performance. In contrast, the value of m does significantly impact on the training time due to the requirement of averaging of m component gradients (i.e., gradients at m recent data points) in Eq. (6). In practice, we suggest to set m to 100.

In the second experiment, we vary n to observe the variation of both accuracy and training time. From Fig. 4, we conclude that the increase of n does slightly impact on

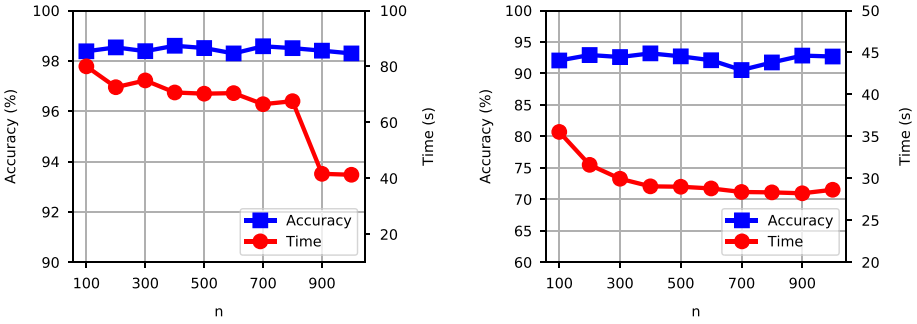

Fig. 4 The variation of accuracy and training time when varying n on the *w8a* (left) and *cod-rna* (right) datasets

the predictive performance, whilst fairly reducing the training time. The reason is that the increase of n reduces the frequency of re-updating $\boldsymbol{u}^d$ (cf. line 10 of Algorithm 1). In contrast, the increase of n leads to a consumption of more main memory using to store the previous models $\mathbf{w}_{t-n+1}, \ldots, \mathbf{w}_t$. Furthermore, according to our theory, a larger value for n leads to a smaller value for ρ, hence resulting in a faster convergence. Therefore, in practice we recommend to set n as large as possible depending on the available memory of learning system. In addition, we empirically found that $n = 100$ works well for real datasets.

5 Conclusion, limitation, and future work

In this paper, we have proposed the *Dual Space Stochastic Variance-reduced Gradient Descent* for Kernel Online Learning(DualSVRG), which obtains the ϵ-approximate *linear* convergence rate and avoids the curse of kernelization. The proposed method is developed relying on the spirit of the variance reduction technique and the idea of dual space. Furthermore, we have also proposed three stochastic oracles that efficiently govern how to store model in a dual space. We have conducted extensive experiments on bench-marked datasets in both batch and online modes. The experimental results have shown that our proposed DualSVRG has achieved superior predictive performance due to its ability in rapidly and accurately solving its optimization problem, whilst spending comparable training time in comparison with its rivals.

Our work has some limitation from the practical aspect and some left questions to answer from the theoretical aspect, which open rooms for future advancements. First, although the time to process one data instance is tolerable, the total training time is generally higher than the baselines. This hints a question to answer in future work: how to approximate relevant gradients more rapidly, while maintaining the approximation precision. From the theoretical perspective, the question regarding the expected variance of gradient approximation is still left unanswered. Additionally, the influence of increasing the budget size B in the budget oracle and the approximation threshold θ in the coverage oracle to the gap between the approximation and true decision functions needs to be addressed more rigorously.

Funding None.

Data availability Yes, source code and data are available.

Declarations

Conflict of interest The authors declares that they have no conflict of interest.

Consent to participate Yes, we agree to participate

Consent for publication Yes, we agree to publish.

References

Bochner, S. (2003). *Lectures on fourier integrals: With an author's supplement on monotonic functions, stieltjes integrals, and harmonic analysis*. Textbook Publishers.

Cavallanti, G., Cesa-Bianchi, N., & Gentile, C. (2007). Tracking the best hyperplane with a simple budget perceptron. *Machine Learning, 69*(2–3), 143–167.

Chang, C.-C., & Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology, 2*(3), 27:1-27:27.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research, 7,* 551–585.

Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2005). The forgetron: A kernel-based perceptron on a fixed budget. In *Advances in Neural Information Processing Systems*, pages 259–266.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning, 37*(3), 277–296.

Hazan, E., Agarwal, A., & Kale, S. (2007). Logarithmic regret algorithms for online convex optimization. *Machine Learning, 69*(2), 169–192.

Hazan, E., & Kale, S. (2014). Beyond the regret minimization barrier: Optimal algorithms for stochastic strongly-convex optimization. *Journal of Machine Learning Research, 15*(1), 2489–2512.

Hensman, J., Fusi, N., & Lawrence, N. D. (2013). Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, 282. Citeseer.

Johnson, R., & Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing (NIPS)*, pages 315–323.

Kivinen, J., Smola, A. J., & Williamson, R. C. (2004). Online learning with Kernels. *IEEE Transactions on Signal Processing, 52,* 2165–2176.

Le, T., Duong, P., Dinh, M., Nguyen, D. T., Nguyen, V., & Phung, D. (2016a). Budgeted semi-supervised support vector machine. In *The 32th Conference on Uncertainty in Artificial Intelligence*, 2016.

Le, T., Nguyen, V., Nguyen, T. D., & Phung, Dinh. (2016b). Nonparametric budgeted stochastic gradient descent. In *The 19th International Conference on Artificial Intelligence and Statistics*, pages 654–572.

Le, T., Nguyen, T. D., Nguyen, V., & Phung, D. (2016). Dual space gradient descent for online learning. In *Advances in Neural Information Processing (NIPS)*.

Lin, X., & Tong, Z. (2014). A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization, 24*(4), 2057–2075.

Lu, J., Hoi, S. C. H., Wang, J., Zhao, P., & Liu, Z.-Y. (2015). Large scale online kernel learning. *Journal of Machine Learning Research, 17,* 1.

Ming, L., Shifeng, W., & Changshui, Z. (2014). On the sample complexity of random fourier features for online learning: How many random fourier features do we need? *The ACM Transactions on Knowledge Discovery from Data, 8*(3), 1–19.

Orabona, F., Keshet, J., & Caputo, B. (2009). Bounded kernel-based online learning. *Journal of Machine Learning Research, 10,* 2643–2666.

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. In *In Neural Infomration Processing Systems*.

Rakhlin, A., Shamir, O., & Sridharan, K. (2012). Making gradient descent optimal for strongly convex stochastic optimization. In *International Conference on Machine Learning (ICML-12)*, pages 449–456.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386–408.

Schmidt, M. W., Roux, N. L., & Bach, F. R. (2013). Minimizing finite sums with the stochastic average gradient. *CoRR*.

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on machine learning*, pages 807–814, New York, NY, USA, ACM.

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

Shalev-Shwartz, S., & Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research, 14*(1), 567–599.

Steinwart, I. (2003). Sparseness of support vector machines. *Journal of Machine Learning Research, 4,* 1071–1105.

Wang, Z., Crammer, K., & Vucetic, S. (2012). Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *Journal of Machine Learning Research, 13*(1), 3103–3131.

Zhang, K., Lan, L., Wang, Z., & Moerchen, F. (2012). Scaling up kernel svm on limited resources: A low-rank linearization approach. In *International conference on artificial intelligence and statistics*, 1425–1434.

Zhao, P., Wang, J., Wu, P., Jin, R., & Hoi, S. C. H. (2012). Fast bounded online gradient descent algorithms for scalable kernel-based online learning. *CoRR*.