Triply Stochastic Gradients on Multiple Kernel Learning

Xiang Li[*], **Bin Gu**[‡], **Shuang Ao**[*], **Huaimin Wang**[†], **Charles X. Ling**[*]
[*]Computer Science Department, University of Western Ontario, Canada
[†]School of Computer, National University of Defense Technology, China
[‡]Nanjing University of Information Science and Technology, China
{lxiang2, sao, charles.ling}@uwo.ca, jsgubin@nuist.edu.cn, hmwang@nudt.edu.cn

Abstract

Multiple Kernel Learning (MKL) is highly useful for learning complex data with multiple cues or representations. However, MKL is known to have poor scalability because of the expensive kernel computation. Dai et al (2014) proposed to use a doubly Stochastic Gradient Descent algorithm (doubly SGD) to greatly improve the scalability of kernel methods. However, the algorithm is not suitable for MKL because it cannot learn the kernel weights. In this paper, we provide a novel extension to doubly SGD for MKL so that both the decision functions and the kernel weights can be learned simultaneously. To achieve this, we develop the triply Stochastic Gradient Descent (triply SGD) algorithm which involves three sources of randomness – the data points, the random features, and the kernels, which was not considered in previous work. We prove that our algorithm enjoys similar convergence rate as that of doubly SGD. Comparing to several traditional MKL solutions, we show that our method has faster convergence speed and achieved better accuracy. Most importantly, our method makes it possible to learn MKL problems with millions of data points on a normal desktop PC.

1 INTRODUCTION

Kernel methods provide a principled way of learning with non-linear representations. However, finding the best kernel for a specific task is not always easy. Meanwhile, many applications may have several candidate data representations, such as learning with multi-modal or multi-source data. In either situation, we may need the learning system to find an optimal combination of several base kernels or data representations. To serve this purpose, Multiple Kernel Learning algorithms (Lanckriet et al, 2004) have been proposed to automatically learn the decision functions as well as the weights for combining the base kernels.

In theory, there are infinite ways of combining kernels, most MKL methods focus on linear kernel combination for efficiency. To prevent overfitting, l_1 and $l_p (p > 1)$ are commonly used norms to regularize the kernel weights. In this paper, we focus on linear kernel combination and l_p-normalization because of its many successful applications and its strong convexity which greatly benefits optimization. To learn a good kernel combination under l_p regularization, many optimization strategies have been proposed. For example, (1). directly solving the problem using Sequential Minimal Optimization (Sun et al, 2010); (2). alternating optimization methods (Kloft et al, 2011) and (3). online learning approaches which process one training instance at a time (Orabona and Jie, 2011). See Bucak et al (2014); Gönen and Alpaydın (2011) for a survey.

In the big data era, machine learning systems often have to deal with very large datasets. However, existing MKL algorithms require computing all the kernel matrices beforehand, which severely hinders scalability: the computation and storage cost is $O(m \cdot l^2)$, where l is the number of data instances and m is the number of kernels. To improve efficiency, random feature methods (Rahimi et al, 2007) have been proposed to approximate a kernel matrix using explicit feature mappings in a lower-dimensional space. However, to achieve accurate solution, the number of random features has to be $O(l)$ even for a single kernel (Rahimi and Recht, 2009). The indication is that the random feature matrix is still at the order of $O(l^2)$, which is hardly scalable.

Recently, doubly stochastic gradients (doubly SGD) (Dai et al, 2014; Xie et al, 2015) have been proposed which sample training points and random features stochastically and update the model parameters according to the functional gradient in the RKHS (Reproducing Kernels Hilbert Space) induced by the kernel function. These methods show strong empirical performance on large-scale classification tasks which are dominated by Deep Neural Networks. To deal with multiple kernels, doubly SGD methods already support the concatenation of random features from several kernels, which is equivalent to learning with the summation of base kernels with fixed weights. However, fixing the kernel weights may not lead to the best classification performance. One of the key advantages of Multiple Kernel Learning is to learn the optimal combination of base kernels.

In this paper, we extend the doubly SGD algorithm for the l_p-MKL setting. The problem is more complicated than the original one considered by doubly SGD: l_p-MKL problem requires learning not only the decision functions but also the optimal weights of kernel combination.

To achieve this, we develop the triply Stochastic Gradient Descent (triply SGD) algorithm whose basic idea is to insert a third level of randomness to the stochastic optimization process. Specifically, we randomly select a kernel, a data point and a set of random features (of the corresponding kernel) in each SGD iteration. This triple-stochastic process was not considered in previous work.

We theoretically demonstrate how such an algorithmic design together with a proper learning rate can lead to an optimization program with $O(1/t)$ convergence rate, where t is the number of iterations. In other words, our algorithm achieves the same order of convergence rate as doubly SGD. As demonstrated in our experiments, triply SGD has faster convergence rate and achieved better accuracy on various classification tasks compared to traditional MKL solutions.

This paper makes two major contributions:

1. We extend the theoretic framework of doubly SGD to learn not only the decision functions but also the weights of kernel combination.

2. We provide an online algorithm that can learn MKL problems with millions of data points on a normal desktop PC.

The rest of the paper is organized as follows. We first review the technical background of l_p Multiple Kernel Learning and doubly stochastic gradient methods. We provide our algorithm in Section 4 and the theoretic analysis in Section 5. In Section 6, we conduct extensive experiments to demonstrate the advantage of our algorithm over traditional MKL methods. The last section concludes the paper.

2 PRELIMINARIES

2.1 MULTIPLE KERNEL LEARNING

In this paper, we focus on the MKL problem. Suppose we have m kernels, in general, MKL optimizes the following goal

$$\min_{\{\beta,\alpha\}} \mathbf{E}(Y, \alpha \sum_{r=1}^{m} \beta_r K_r) + \mathbf{R}(\alpha, K') \tag{1}$$

where α are the dual coefficients, β are the kernel weights. If we use a Lipschitz loss function $l(\cdot)$ in the error term $\mathbf{E}(\cdot)$ and l_2 norm in the regularizer $\mathbf{R}(\cdot)$, Eq. (1) is equivalent to

$$\min_{\{f,\beta\}} R_{\text{MKL}}(f,\beta) :=$$

$$\mathbb{E}_{(x,y)}\left[l(\sum_{r=1}^{m} f_r(x), y)\right] + \frac{\nu}{2}\sum_{r=1}^{m} \beta_r ||f_r||^2_{\mathcal{H}_r}; \tag{2}$$

where $\mathcal{H}_r$ is the RKHS induced by kernel K_r and f_r is a functional in space $\mathcal{H}_r$.

In particular, the l_p-MKL ($p > 1$) problem (Kloft et al, 2011) constrains the kernel weights to be within the p-norm simplex $\Delta_p = \{||\beta||_p < 1, \beta_r > 0\}$. Under this constraint, Xu et al (2010) have shown that (2) is equivalent to

$$\min_{\{f,\beta\}} R_{l_p\text{MKL}}(f,\beta)$$

$$:= \mathbb{E}_{(x,y)}\left[l(\sum_{r=1}^{m} f_r(x), y)\right] + \frac{\nu}{2}\sum_{r=1}^{m} \frac{||f_r||^2_{\mathcal{H}_r}}{\beta_r};$$

$$s.t., \beta \in \Delta_p, p > 1 \tag{3}$$

This problem can be solved by alternative optimization approaches (Xu et al, 2010; Kloft et al, 2011) which can exploit off-the-shelf SVM solvers in each iteration. l_p MKL can also be optimized using online learning approaches (Orabona and Jie, 2011; Orabona et al, 2012) that use stochastic optimization to process the data one point at a time. However, all these traditional l_p MKL solutions require computing and storing all the m kernels, which sets up obstacles for scaling-up.

2.2 RANDOM FEATURE APPROXIMATION

There have been many attempts to make kernel methods scalable, such as the Nyström method (Williams and Seeger, 2001), divide-and-conquer SVM solver (Hsieh et al, 2014) and budget online kernel learning (Lu et al, 2015). In this paper, we focus on the random feature method (Rahimi et al, 2007), which explicitly maps the decision function $f(\cdot)$ to a lower dimensional space through random processes.

By Bochner's Theorem, for any PSD (Positive Semi-Definitive) kernel $k(\cdot,\cdot)$, there exists a set Ω, a probability measure $\mathbb{P}$ and a random feature $\phi_\omega(x)$, such that

$$k(x,x') = \int_\Omega \phi_\omega(x)\phi_\omega(x')d\mathbb{P}(\omega) \tag{4}$$

Random feature approximation methods explicitly compute $\phi_\omega(x)$ for many kernel functions such as RBF kernels (Rahimi et al, 2007), dot-product kernels (Kar and Karnick, 2012), Laplacian kernels (Yang et al, 2014), to name a few. They approximate a kernel using

$$k(x,x') \approx \frac{1}{D}\sum_{j=1}^{D}\phi_{\omega_j}(x)\phi_{\omega_j}(x') \tag{5}$$

where D is the number of random features for each instance. It requires $O(lDd)$ computation and $O(lD)$ storage to generate random features for a dataset of l instances. Subsequent algorithm can do efficient linear classification on the generated random features:

$$Z(x) = [\phi_{\omega_1}(x)\ \phi_{\omega_2}(x)\ \ldots\ \phi_{\omega_D}(x)] \tag{6}$$

Unfortunately, previous studies (Rahimi and Recht, 2009) have shown that we need $D = O(l)$ to get accurate predictive performance, which implies that the random feature matrix still needs to be $O(l^2)$.

2.3 DOUBLY STOCHASTIC FUNCTIONAL GRADIENTS

To further improve the scalability of random feature methods, Dai et al (2014) have proposed the Doubly Stochastic Functional Gradient (Doubly SGD) algorithm which applies classical SGD procedure to stochastic random feature learning. Specifically, whenever the SGD procedure receives a new data point (or a batch of points), doubly SGD will also generate its corresponding random features on-the-fly, and compute the stochastically-approximated functional gradient to update the model parameters. For the prediction purpose, the system only needs to store the seeds used for generating the random features, which leads to great improvement in scalability. Algorithm 1 and 2 describe the doubly SGD procedure. On the theoretic side, Dai et al (2014) have proved that the doubly SGD algorithm can achieve $O(1/t)$ convergence rate under certain reasonable assumptions.

Notice that the doubly SGD algorithm can be used for multidimensional random features, i.e., $\phi_\omega(x) = [\phi_{\omega_1}(x)\phi_{\omega_2}(x),\ldots,\phi_{\omega_m}(x)] \in \mathbb{R}^m$, and each dimension r corresponds to a different kernel $k_r(\cdot,\cdot)$. This formulation is actually equivalent to Multiple Kernel Learning with fixed kernel weights, i.e., Eq. (2) with $\beta = \mathbf{1}$. However, fixed kernel weights unavoidably compromises model expressiveness, and the generalization performance may suffer especially when we are dealing with large datasets. Meanwhile, directly applying doubly SGD to MKL means that we have to generate random features for all the m kernels in each iteration, this could hinder the scalability especially when the number of kernels m is large.

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

```
for $i = 1,\ldots,t$ do
    sample $(x_i, y_i)$;
    sample $\omega_i \sim \mathbb{P}(\omega)$ seed $i$;
    $f(x_i)$ = DoublySgdPredict($x_i, \{\alpha_j\}_{j=1}^{i-1}$);
    $\gamma_i = \frac{\theta}{i}$;
    $\alpha_i = -\gamma_i l'(f(x_i), y_i)\phi_{\omega_i}(x_i)$;
    $\alpha_j = (1-\gamma_i\nu)\alpha_j$ for $j = 1,\ldots,i-1$;
end for
```

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

```
set $f(x) = 0$
for $i = 1,\ldots,t$ do
    sample $\omega_i \sim \mathbb{P}(\omega)$ seed $i$;
    $f(x) = f(x) + \alpha_i\phi_{\omega_i}(x)$;
end for
```

In order to address these challenges, we propose the Triply Stochastic Functional Gradient method (triply SGD), which can learn the kernel weights as well as the decision functions. Most importantly, it only requires generating random features for one stochastically selected kernel in each SGD update, which also opens the door for developing parallel implementations.

3 STOCHASTIC FUNCTIONAL GRADIENTS FOR l_p MKL

It is known that the convex combination of PSD kernels is also a PSD kernel. The RKHS $\mathcal{H}$ in MKL learning can be constructed by the sum space of functions (Sindhwani and Rosenberg, 2008). i.e.,

$$\mathcal{H} := \{\mathcal{H}_1 \oplus \mathcal{H}_2 \oplus \ldots \oplus \mathcal{H}_m\}$$

$$:= \{F | F(x) = \sum_{r=1}^{m} f_r(x), \forall f_r \in \mathcal{H}_r\} \tag{7}$$

For the RKHS $\mathcal{H}$ defined by (7), we always have

$$\sum f_r(x) \in \mathcal{H} \tag{8}$$

$$||\sum f_r||^2_{\mathcal{H}} = \sum ||f_r||^2_{\mathcal{H}_r} \tag{9}$$

$$\nabla(\sum f_r) = \sum(\nabla f_r) \tag{10}$$

From Xu et al (2010), we know that l_p MKL problem (3) is equivalent to

$$\min_{\{f_r \in \mathcal{H}_r\}_{r=1}^m} \mathbb{E}_{(x,y)}\Big[l(F(x), y)\Big] + \frac{\nu}{2} \cdot \Psi \tag{11}$$

where

$$F(x) := \sum_{r=1}^m f_r(x)$$

$$\Psi := (\sum_{r=1}^m ||f_r||^q_{\mathcal{H}_r})^{2/q}$$

and $q = \frac{2p}{p+1}$. From $p > 1$, we know that $1 < q < 2$. This leads to a $\frac{\nu(p-1)}{2p}$-strongly convex problem (Orabona et al, 2012). [1]

Using this formulation, the functional gradient w.r.t. one of the decision functions would be

$$\nabla R(f_r) = \mathbb{E}_{(x,y)}[\xi_r(\cdot)] + \frac{\nu}{2}\nabla\Psi(f_r) \tag{12}$$

where $\nabla\Psi(f_r)$ is the functional gradient due to the regularization term and $\xi_r(\cdot) = l'(F(x), y)k_r(x, \cdot)$ is that of the error term. Since we use stochastically generated random features, we cannot compute the decision functional $k_r(x, \cdot)$ explicitly. Instead, we compute the *stochastic* functional gradient with respect to each decision function f_r:

$$\zeta_r(\cdot) = l'(F(x), y)\phi_{\omega_r}(x)\phi_{\omega_r}(\cdot)$$

which satisfies $\xi_r(\cdot) = \mathbb{E}_{\omega_r}[\zeta_r(\cdot)]$.

Using some basic algebraic transformation, $\nabla\Psi(f_r)$ can be represented as:

$$\nabla\Psi(f_r) = 2f_r\psi_r \tag{13}$$

where we define

$$\psi_r := \Big[\frac{\sum_{s=1}^m ||f_s||^q_{\mathcal{H}_s}}{||f_r||^q_{\mathcal{H}_r}}\Big]^{\frac{2-q}{q}} \tag{14}$$

In this paper, we will use the convention that $\frac{t}{0} = 0$ when $t = 0$ and $+\infty$ otherwise (Kloft et al, 2011).

So far we have derived the stochastic functional gradients w.r.t. each individual decision function of the l_p MKL problem. We next develop the triply SGD algorithm based on these stochastic gradients.

[1]Equivalently, $\frac{\nu(q-1)}{q}$-strongly convex.

4 TRIPLY STOCHASTIC GRADIENT DESCENT

The main idea of the triply SGD algorithm is to optimize the l_p MKL objective (11) with three levels of stochastic samplings: data points, kernels and random features; and update the model using stochastic functional gradients with respect to each decision function. The pseudo code is given in Algorithms 3 and 4, where $\phi_{\omega_{r,j}}$ and $\alpha_{r,j}$ denote the random feature and dual coefficient for kernel r sampled in iteration j, respectively. Agian, compared to the multi-kernel extension of the original doubly SGD, triply SGD only needs to sample one kernel in each iteration, which makes it more efficient in memory usage.

Algorithm 3 $\{\alpha_{1:r,i}\}_{i=1}^t$ =**TriplySGD**($\mathbb{P}(x, y), \theta$)

```
1:  for $i = 1, \dots, t$ do
2:      sample $(x_i, y_i)$;
3:      sample kernel $r_i \sim 1 \dots m$ seed $i$;
4:      sample $\omega_{r_i,i} \sim \mathbb{P}_{r_i}(\omega)$ seed $i$;
5:      $\{f_r(x_i)\}_{r=1}^m$ = Predict($x_i, \{\alpha_{1:r,j}\}_{j=1}^{i-1}$);
6:      $\psi_i = \Big[\frac{\sum_{s=1}^m ||f_s||^q_{\mathcal{H}_s}}{||f_r||^q_{\mathcal{H}_r}}\Big]^{\frac{2-q}{q}}$;
7:      $\gamma_i = \frac{\theta}{i}$;
8:      $\eta_i = \frac{\gamma_i}{\psi_i}$;
9:      $\alpha_{r_i,i} = -\eta_i l'(\sum_{r=1}^m f_r(x_i), y_i)\phi_{\omega_{r_i,i}}(x_i)$;
10:     $\forall r, \alpha_{r,j} = (1 - \gamma_i\nu)\alpha_{r,j}$ for $j = 1, \dots, i-1$;
11: end for
```

Algorithm 4 $\{f_r(x)\}_{r=1}^m$ =**Predict**($x, \{\alpha_{1:r,i}\}_{i=1}^t$)

```
1:  set $f_1(x) = f_2(x) = \dots = f_m(x) = 0$
2:  for $i = 1, \dots, t$ do
3:      sample kernel $r_i \sim 1 \dots m$ seed $i$;
4:      sample $\omega_{r_i,i} \sim \mathbb{P}_{r_i}(\omega)$ seed $i$;
5:      $f_{r_i}(x) = f_{r_i}(x) + \alpha_{r_i,i}\phi_{\omega_{r_i,i}}(x)$;
6:  end for
```

Next we illustrate how lines 8-10 of the algorithm update the functional of each kernel K_r and the global functional F.

At iteration t, denote r_t as the chosen kernel, $\mathcal{D}^t = (x_t, y_t)$ as the selected data point and $f_{r,t}$ as the decision function of kernel r. For brevity, we use ψ_t to denote ψ_{r_t}. We have $f_{r,1}(\cdot) = 0$ and

$$f_{r,t+1}(\cdot) = $$

$$\begin{cases} f_{r,t}(\cdot) - \nu\gamma_t f_{r,t}(\cdot) - \eta_t\zeta_{r,t}(\cdot) & \text{if } r_t = r \\ f_{r,t}(\cdot) - \nu\gamma_t f_{r,t}(\cdot) & \text{else} \end{cases} \tag{15}$$

$$F_{t+1}(\cdot) = F_t(\cdot) - \nu\gamma_t F_t(\cdot) - \eta_t\zeta_{r_t,t}(\cdot) \tag{16}$$

In other words, since we only sampled one kernel at a time, the gradient $\zeta_{r_t,t}(\cdot)$ will only be used to update the functional of that kernel r_t. All other functionals ($f_r, r \neq r_t$) will only shrink by a constant factor $(1-\nu\gamma_t)$.

Meanwhile, since $f_r(\cdot)$ and $F(\cdot)$ may not be in the corresponding RKHS $\mathcal{H}_r$ and $\mathcal{H}$ during the stochastic learning process, we will also construct intermediate functionals $h_r(\cdot)$ and $H(\cdot)$ as did in Dai et al (2014). Accordingly, $h_{r,1}(\cdot) = 0$ and

$$h_{r,t+1}(\cdot) = \begin{cases} h_{r,t}(\cdot) - \nu\gamma_t h_{r,t}(\cdot) - \hat{\eta}_t \xi_{r,t}(\cdot) & \text{if } r_t = r \\ h_{r,t}(\cdot) - \nu\gamma_t h_{r,t}(\cdot) & \text{else} \end{cases} \quad (17)$$

$$H_{t+1}(\cdot) = H_t(\cdot) - \nu\gamma_t H_t(\cdot) - \hat{\eta}_t \xi_{r_t,t}(\cdot) \quad (18)$$

where we have set $\hat{\eta}_t = \frac{\gamma_t}{\hat{\psi}_t}$, in which

$$\hat{\psi}_t := \Big[\frac{\sum_{s=1}^m ||h_{s,t}||^q_{\mathcal{H}_s}}{||h_{r,t}||^q_{\mathcal{H}_{r_t}}}\Big]^{\frac{2-q}{q}} \quad (19)$$

corresponds to Eq. (14).

Since the kernels are stochastically selected, for ease of illustration, we further denote the following auxiliary variables:

(1). $I^i(r) := \{j | r_j = r\}$, all iterations (up to i) where kernel r gets selected;

(2). $T^i(r) := \max\{I^i(r)\}$, the most recent iteration (up to i) that kernel r gets selected;.

(3). $C^i(r) := |I^i(r)|$, the number of iterations (up to i) that kernel r gets selected.

By these definitions, $T^i(r_i) = i$ and $C^{T^i(r)}(r) = C^i(r)$ always true. We can rewrite (15)(17) as

$$f_{r,t+1}(\cdot) = \sum_{i \in I^t(r)} u^i_t(r) \zeta_{r,i}(\cdot) \quad (20)$$

$$h_{r,t+1}(\cdot) = \sum_{i \in I^t(r)} \hat{u}^i_t(r) \xi_{r,i}(\cdot) \quad (21)$$

where

$$u^i_t(r) = -\eta_{T^i(r)} \prod_{j=T^i(r)+1}^{t} (1-\nu\gamma_j) \quad (22)$$

$$\hat{u}^i_t(r) = -\hat{\eta}_{T^i(r)} \prod_{j=T^i(r)+1}^{t} (1-\nu\gamma_j) \quad (23)$$

Notice that $u^i_t(r), \hat{u}^i_t(r)$ are indeterministic sequences, which depend on the orders of kernel sampling. However, if we look at the global functionals $F(\cdot), H(\cdot)$, we can rewrite (16)(18) as

$$F_{t+1}(\cdot) = \sum_{i=1}^{t} v^i_t \zeta_{r_i,i}(\cdot) \quad (24)$$

$$H_{t+1}(\cdot) = \sum_{i=1}^{t} \hat{v}^i_t \xi_{r_i,i}(\cdot) \quad (25)$$

where

$$v^i_t = -\eta_i \prod_{j=i+1}^{t} (1-\nu\gamma_j) = u^i_t(r_i) \quad (26)$$

$$\hat{v}^i_t = -\hat{\eta}_i \prod_{j=i+1}^{t} (1-\nu\gamma_j) = \hat{u}^i_t(r_i) \quad (27)$$

are deterministic sequences.

Notice that

$$v^i_t = -\frac{a^i_t}{\psi_i}; \;\; \hat{v}^i_t = -\frac{a^i_t}{\hat{\psi}_i}$$

where $a^i_t = -\gamma_i \prod_{j=i+1}^t (1-\nu\gamma_j)$ is the same as defined in Dai et al (2014), with step size $\gamma_t = \frac{\theta}{t}$. These settings help us connect to the theoretic framework of Dai et al (2014).

5 ANALYSIS

In this section, we analyze Algorithm 3 and 4 under the following practical assumptions:

A1. Both the loss function $l(u, y)$ and its first-derivative $l'(u, y)$ are L-Lipschitz continuous.

A2. For any data $\{(x_i, y_i)\}_{i=1}^t$ and any sequence of functionals $\{F_i\}_{i=1}^t$, the first-derivative of loss is bounded: $l'(F_i(x_i), y_i) \leq M$.

A3. There exists $\kappa > 0$ and $\phi > 0$, such that for any kernel $r \in \{1, \ldots, m\}$, $\forall x, \omega$: $|k_r(x, x')| \leq \kappa$, $|\phi_\omega(x)\phi_\omega(x')| \leq \phi$.

A4. There exists $C_q > 1$, such that, $\forall \{f_r\}_{r=1}^m$

$$1 \leq \Big[\frac{\sum_{s=1}^m ||f_s||^q_{\mathcal{H}_s}}{||f_r||^q_{\mathcal{H}_r}}\Big]^{\frac{2-q}{q}} \leq C_q$$

For A4, the first inequality always holds because $q \in (1,2)$. It implies that $\hat{\psi}_t \in [1, C_q]$ and $\psi_t \in [1, C_q]$.

Under the above conditions, we are able to prove the following theorem, which implies that the algorithm has $O(1/t)$ convergence rate.

Theorem 5.1 (Convergence in Expectation) *Set* $\gamma_i = \frac{\theta}{i}(1 \leq i \leq t)$, $\theta > 0$ *and* $\theta\nu \in (1,2) \cup \mathbb{Z}_+$ *for Algorithm 3 and 4, we always have*

$$\mathbb{E}_{\mathcal{D}^t, r^t, \omega^t}\Big[|F_{t+1}(x) - F_*(x)|^2\Big] \leq \frac{2C^2 + 2\kappa Q_1^2}{t} \quad (28)$$

Table 1: Dataset information and experiment settings. In the experiments of Figure 1, we only use a subset of training data as indicated in the 5th column of this table.

data	classes	PCA dim	total size	subset size	kernels	kernel size	λ
covtype	2	40	581,012	20,000	5 rbf	22 GB	10^{-5}
epsilon	2	100	400,000	20,000	5 rbf	22 GB	10^{-5}
mnist8m (binary)	2	100	1,625,000	20,000	5 rbf	22 GB	10^{-5}
cifar10 (binary)	2	100	50,000	20,000	5 rbf	22 GB	10^{-5}
mnist8m	10	100	8,100,000	20,000	5 rbf	22 GB	10^{-5}
cifar10	10	100	50,000	25,000	3 rbf	15 GB	10^{-5}

where we define $Q_1 = \max\{||F_*||_{\mathcal{H}}, \frac{Q_0+\sqrt{Q_0^2+(2\nu'\theta-1)\kappa M^2(1+\nu\theta)^2(\theta)^2}}{(2\nu'\theta-1)}\}$, *and* $Q_0 = 2\sqrt{2}\kappa^{1/2}LM(\kappa+\phi)\theta^2$, $C = 4(\kappa+\phi)^2M^2\theta^2$ *and* $\nu' = \frac{\nu(p-1)}{2pC_q}$.

Remarks. Notice that $F_*(\cdot)$ is the optimal functional in the RKHS for solving the l_p MKL problem (11). The above theorem implies that after $t+1$ iterations the functional found by triply SGD will converge to $F_*(\cdot)$, and the difference is no more than $\frac{2C^2+2\kappa Q_1^2}{t}$.

Proof: (sketch) The basic flow of the proof follows that of doubly SGD. In other words, we also decompose the errors into the error due to random features and the error due to random data, i.e.,

$$|F_{t+1}(x) - F_*(x)|^2 \leq 2|F_{t+1}(x) - H_{t+1}(x)|^2 + 2\kappa||H_{t+1} - F_*||_{\mathcal{H}} \tag{29}$$

we use Lemma 5.2 and 5.3 to bound each error term respectively. Detailed proofs are deferred to the appendix. □

Lemma 5.2 *For any* $x \in \mathcal{X}$,

$$\mathbb{E}_{\mathcal{D}^t,r^t,\omega^t}\left[\left|F_{t+1}(x) - H_{t+1}(x)\right|^2\right] \leq B_{1,t+1}^2 \tag{30}$$

where $B_{1,t+1}^2 := M^2(\kappa+\phi)^2\sum_{i=1}^t |a_t^i|^2$.

Lemma 5.3 *Set* $\gamma_i = \frac{\theta}{i}(1 \leq i \leq t)$, $\theta > 0$ *and* $\theta\nu \in (1,2) \cup \mathbb{Z}_+$, *we have*

$$\mathbb{E}_{\mathcal{D}^t,r^t,\omega^t}\left[||H_{t+1} - F_*||_{\mathcal{H}}^2\right] \leq \frac{Q_1^2}{t} \tag{31}$$

where Q_1 *is as defined in Theorem 5.1.*

In the next section, we use experiments to demonstrate the effectiveness and efficiency of triply SGD on l_p MKL tasks.

6 EXPERIMENTS

In this section, we compare the proposed triply SGD algorithm to several traditional MKL methods with various experiments.

6.1 BASELINES

Among the many traditional MKL methods, we select OBSCURE (Orabona et al, 2012), UFO (Orabona and Jie, 2011) and p-norm MKL (Kloft et al, 2011) as the baselines in this paper. OBSCURE and UFO are online algorithms that can directly optimize the l_p-MKL goal using stochastic sub-gradient descent (Shalev-Shwartz et al, 2011) while the p-norm MKL algorithm optimizes kernel weights β and the SVM problem with fixed β alternatively. They are highly competitive MKL solvers in terms of convergence speed and accuracy according to extensive experiments on medium size datasets. (Orabona et al, 2012; Orabona and Jie, 2011; Kloft et al, 2011). Besides, in order to show the benefit of MKL and why doubly SGD is not enough, we include the straightforward extension of doubly SGD as a baseline. To make doubly SGD compatible with the MKL scenario, we generate and concatenate the random features from all the kernels in each iteration of Algorithm 1 and 2. We denote this naive extension as *DoublySGD-MK*.

6.2 DATASETS

The purpose of our experiments is to demonstrate the scalability of the proposed method. For this purpose, we choose large scale datasets on which traditional MKL algorithms are hardly scalable. For binary classification tasks, we have used the *epsilon*, *covtype*, *mnist8m-binary* from the LibSVM DATA collection[2] and the *cifar10-binary* (Krizhevsky et al, 2012) dataset. The *cifar10-binary* task requires to classify whether an image is animal or not, *mnist8m-binary* classifies the digits '6' and '8'. For multi-class problems, we consider all the 10 classes in *cifar10* and *mnist8m*. The dataset information

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

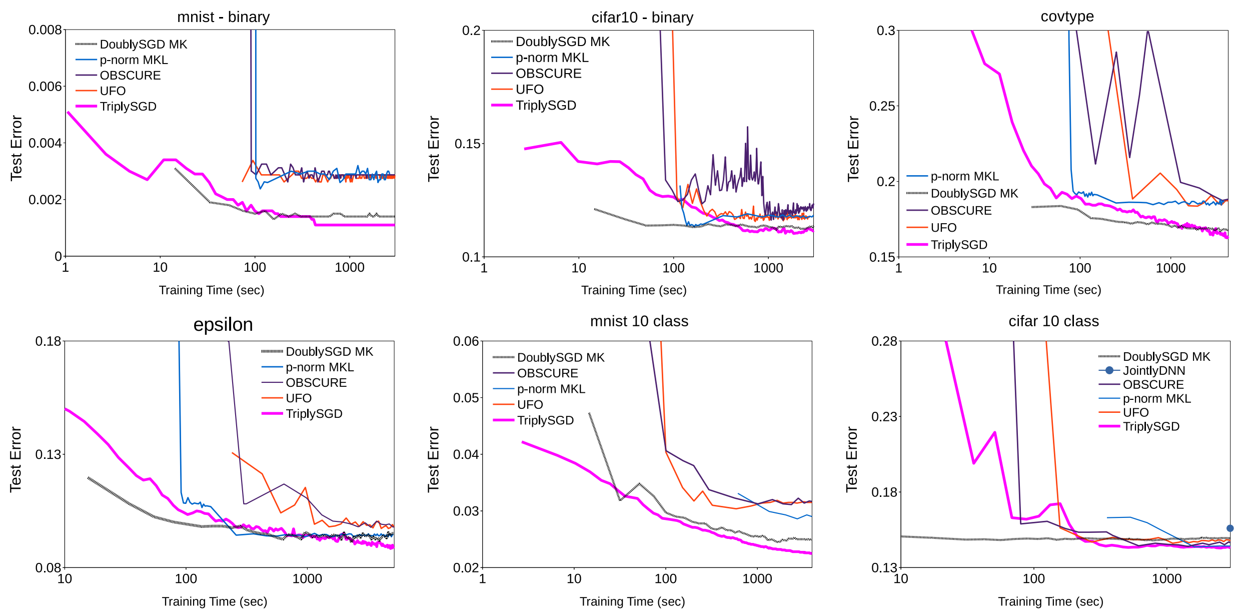

Figure 1: Learning curves on datasets with fixed training sizes. Experiment settings are given in Table 1 and Section 6.3. Best performance of the CNN we used for feature extraction is also indicated as a blue circle. Notice that the curves of traditional MKL solvers start at around 100 seconds, because they need to compute the kernel matrices before training. *Compared to Triply SGD, Doubly SGD converges faster at the beginning since it does not need to learn the kernel weights. However, as a result, its generalization error is higher than triply SGD in the long run.*

Table 2: Comparison of the final test error rates between doubly and triply SGD in Figure 1.

	mnist-bin	cifar-bin	covtype-bin	epsilon-bin	cifar-10	mnist-10
DoublySGD-MK	0.0014	0.1190	0.1680	0.095	0.1497	0.0250
TriplySGD	**0.0011**	**0.1120**	**0.1630**	**0.088**	**0.1407**	**0.0220**

and parameter settings are shown in Table 1.

6.3 MULTIPLE KERNELS

In order to build a multi-kernel scenario for the *cifar10-multiclass* dataset, we use 3 sets of CNN-pretrained features to construct three different RBF kernels. The first set of features are extracted from the last dense layer (512 dimensions) of a well-trained 18-layer-ResNet (He et al, 2016) with 15.6% test error rate after 190 training epochs. The other two sets of features are extracted from a lightly-trained (5 epochs of training) CNN (Lin et al, 2013) with 29% test error rate. We use the last dense layer (512 dimensions) and the last max-pooling layer (2304 dimensions), respectively. In this way, we intentionally made one of the kernels more informative than the other two, so that adjusting the kernel weights is important. We apply PCA to reduce the dimensionality of each feature representation to 100 before computing the RBF functions.

For all other datasets, we apply PCA on the original features or raw pixels for dimensionality reduction. We construct 5 kernels on the PCA dimensions. The first kernel is simply computed using all PCA dimensions. The other 4 kernels are constructed from each $\frac{1}{4}$ of the PCA dimensions. In this way, we still intentionally make the first kernel more informative. The bandwidths of all RBF kernels are determined by the median trick.

6.4 IMPLEMENTATION DETAILS

We run all experiments on a desktop computer with an 8 core Intel i7 CPU and 24 gigabyte memories. All algorithms are implemented in MATLAB. For the baseline methods, the computer memory can only accommodate the kernels of a subset of data, which is $20,000/8,000$ (train/test) instances for the five kernel experiments and $25,000/10,000$ instances for the three kernel experiments. For the triply SGD algorithm, we set the number of random features as 2^{13} and the batch size is set as 2^{15}. With this setting, the triply SGD program never exceeds 16 GB memory-usage, irrespective of the training size.

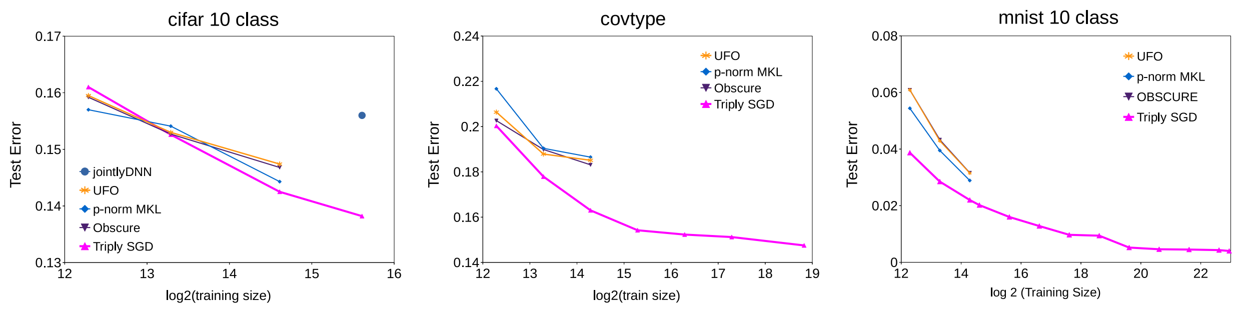

Figure 2: Experiments with different training sizes. The baselines can only handle a small subset of each dataset. For each sample size of a given dataset, we set a time budget of $10,000$ seconds. The other settings including the construction of kernels are the same as Figure 1.

For the parameter p and q in the l_p MKL objective (3), we use $q = \frac{2\log m}{2\log m - 1}$ as suggested in Orabona and Jie (2011).

All the candidate algorithms use hinge loss for the binary classification tasks. To handle multi-class scenarios, the OBSCURE and UFO algorithms have used the 'max-of-hinge' loss (Crammer and Singer, 2001). While the original p-norm MKL approach focuses on binary classification, we use the One-Versus-Rest strategy to extend it to the multi-class setting. For the triply SGD algorithm, we use the multi-class logistic (i.e., softmax) loss as did in Dai et al (2014); Li and Póczos (2016).

6.5 EXPERIMENTS AND RESULTS

We first evaluate the convergence speed and accuracy of all candidate methods with the same training size. In other words, we restrict the triply SGD algorithm to use the same subsets of data as the baselines. See Table 1 for our settings. We use a time budget of $3,000$ seconds for each experiment which includes kernel computation and random feature generation as well as training. Figure 1 depicts the learning curves of test error for different tasks.

Observation 1. *Compared to traditional MKL solvers, Triply SGD has a faster learning convergence speed and achieves higher predictive performance.*

Table 2 shows the final test error of triplySGD and doublySGD-MK (after $3,000$ seconds). In all six experiments, TriplySGD has better performance. This is because doublySGD-MK simply concatenates the random features from multiple kernels and cannot adjust the kernel weights. As a result, this naive extension of doublySGD can hardly take full advantage of the flexibility of Multiple Kernel Learning.

Observation 2. *Compared to Doubly SGD, Triply SGD can achieve better accuracy in the MKL scenario.*

In order to show how triply SGD can handle larger datasets and gain better performance, we conduct experiments with various training sizes using the *covtype*, *mnist8m* and *cifar10* datasets. We keep the same parameter settings and kernel composition as our previous experiments. For baseline methods, they can only process a subset of each data. For the triply SGD algorithm, we are always able to increase the training size until the entire datasets are used. For each sample size of a given dataset, we set a running time budget of $10,000$ second and report the final test accuracy. The results are shown in Figure 2.

Observation 3. *Triply SGD is scalable to large datasets and achieves improved accuracy as the data size increases.*

The experiment on *mnist8m* data also indicates that triply SGD can learn l_p-MKL problem with millions of data points.

7 CONCLUSION

In this paper, we provide a novel extension to doubly SGD for MKL problems so that both the decision functions and the kernel weights can be learned. To achieve this, we develop the triply Stochastic Gradient Descent (triply SGD) algorithm which involves three sources of randomness – the data points, the random features, and the kernels, which was not considered in previous work. We prove that our algorithm has $O(1/t)$ convergence rate. Comparing to several traditional MKL solutions, our experiments show that triply SGD has faster convergence speed and achieved better accuracy. Most importantly, it is possible to learn MKL problems with millions of data points on a normal desktop PC.

Acknowledgments

This work is supported by National Natural Science Foundation of China (grants 61432020, 61472430, 61573191) and Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Bucak S. S., Jin R., Jain A. K. (2014) Multiple kernel learning for visual object recognition: A review. IEEE Transactions on Pattern Analysis and Machine Intelligence 36(7):1354–1369

Crammer K., Singer Y. (2001) On the algorithmic implementation of multiclass kernel-based vector machines. Journal of machine learning research 2(Dec):265–292

Dai B., Xie B., He N., Liang Y., Raj A., Balcan M.-F. F., Song L. (2014) Scalable kernel methods via doubly stochastic gradients. In: Advances in Neural Information Processing Systems, pp. 3041–3049

Gönen M., Alpaydın E. (2011) Multiple kernel learning algorithms. Journal of Machine Learning Research 12(Jul):2211–2268

He K., Zhang X., Ren S., Sun J. (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778

Hsieh C.-J., Si S., Dhillon I. (2014) A divide-and-conquer solver for kernel support vector machines. In: International Conference on Machine Learning, pp. 566–574

Kar P., Karnick H. (2012) Random feature maps for dot product kernels. In: Proceedings of The 15th International Conference on Artificial Intelligence and Statistics(AISTATS)

Kloft M., Brefeld U., Sonnenburg S., Zien A. (2011) Lp-norm multiple kernel learning. Journal of Machine Learning Research 12(Mar):953–997

Krizhevsky A., Sutskever I., Hinton G. E. (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105

Lanckriet G. R., De Bie T., Cristianini N., Jordan M. I., Noble W. S. (2004) A statistical framework for genomic data fusion. Bioinformatics 20(16):2626–2635

Li C.-L., Póczos B. (2016) Utilize old coordinates: faster doubly stochastic gradients for kernel methods. In: Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, AUAI Press, pp. 467–476

Lin M., Chen Q., Yan S. (2013) Network in network. arXiv preprint arXiv:13124400

Lu J., Hoi S. C., Sahoo D., Zhao P. (2015) Budget online multiple kernel learning. arXiv preprint arXiv:151104813

Orabona F., Jie L. (2011) Ultra-fast optimization algorithm for sparse multi kernel learning. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 249–256

Orabona F., Jie L., Caputo B. (2012) Multi kernel learning with online-batch optimization. Journal of Machine Learning Research 13(Feb):227–253

Rahimi A., Recht B. (2009) Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In: Advances in neural information processing systems, pp. 1313–1320

Rahimi A., Recht B., et al (2007) Random features for large-scale kernel machines. In: Advances in Neural Information Processing Systems

Shalev-Shwartz S., Singer Y., Srebro N., Cotter A. (2011) Pegasos: Primal estimated sub-gradient solver for svm. Mathematical programming 127(1):3–30

Sindhwani V., Rosenberg D. S. (2008) An rkhs for multi-view learning and manifold co-regularization. In: Proceedings of the 25th international conference on Machine learning, ACM, pp. 976–983

Sun Z., Ampornpunt N., Varma M., Vishwanathan S. (2010) Multiple kernel learning and the smo algorithm. In: Advances in neural information processing systems, pp. 2361–2369

Williams C. K., Seeger M. (2001) Using the nyström method to speed up kernel machines. In: Advances in neural information processing systems, pp. 682–688

Xie B., Liang Y., Song L. (2015) Scale up nonlinear component analysis with doubly stochastic gradients. In: Advances in Neural Information Processing Systems, pp. 2341–2349

Xu Z., Jin R., Yang H., King I., Lyu M. R. (2010) Simple and efficient multiple kernel learning by group lasso. In: Proceedings of the 27th international conference on machine learning (ICML-10), pp. 1175–1182

Yang J., Sindhwani V., Fan Q., Avron H., Mahoney M. W. (2014) Random laplace feature maps for semigroup kernels on histograms. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 971–978