Large-scale k-means clustering via variance reduction

Yawei Zhao, Yuewei Ming, Xinwang Liu, En Zhu

National University of Defense Technology, Changsha, Hunan, China

Kaikai Zhao

Naval Aeronautical University, Yantai, Shandong, China

Jianping Yin

Dongguan University of Technology, Dongguan, Guangdong, China

Abstract

With the increase of the volume of data such as images in web, it is challenging to perform k-means clustering on millions or even billions of images efficiently. One of the reasons is that k-means requires to use a batch of training data to update cluster centers at every iteration, which is time-consuming. Conventionally, k-means is accelerated by using one or a mini-batch of instances to update the centers, which leads to a bad performance due to the stochastic noise. In the paper, we decrease such stochastic noise, and accelerate k-means by using variance reduction technique. Specifically, we propose a position correction mechanism to correct the drift of the cluster centers, and propose a variance reduced k-means named VRKM. Furthermore, we optimize VRKM by reducing its computational cost, and propose a new variant of the variance reduced k-means named VRKM++. Comparing with VRKM, VRKM++ does not have to compute the batch gradient, and is more efficient. Extensive empirical studies show that our methods VRKM and VRKM++ outperform the state-of-the-art method, and obtain about 2× and 4× speedups for large-scale clustering, respectively. The source code is available at https://github.com/YaweiZhao/VRKM_sofia-ml.

1. Introduction

K-means clustering has been intensively studied and widely applied into various applications, such as image segmentation [1], outlier detection [2], sense discovery [3], to name just a few. With a group of initialized cluster centers, k-means algorithm alternatively performs instance partition and the update of cluster centers until convergence [4]. As seen, it needs to pass over a batch of instances in order to update cluster centers at each iteration, which is computationally intensive. It prevents traditional k-means algorithms from being applied into large scale applications.

To enable the ability of k-means in dealing with large-scale applications, the pioneering work in [5] proposes a stochastic gradient descent (SGD) variant of k-means in which one instance is randomly sampled to update a cluster center at each iteration. Compared with the traditional batch k-means, this variant avoids scanning all samples when updating the cluster centers, and significantly improves the computational efficiency of k-means. However, by randomly sampling, this variant usually brings in stochastic noise or variance[1]. Such stochastic noise impairs the convergence of the objective function, which adversely affects the clustering performance. Recently, a mini-batch variant of k-means is proposed in [6] to alleviate this issue. Instead of sampling one instance at each iteration, it performs the update of cluster centers by randomly sampling a mini-batch instances. As seen, the mini-batch variant of k-means is able to effectively decrease the stochastic noise while increasing the computational cost in calculating the gradient. Some more recently work on SGD such as variance reduction technique (SVRG) [7] has been developed to decrease the stochastic noise. In SVRG, the parameters are updated via the variance reduced gradient whose expectation equals to the batch gradient. By this manner, SVRG is able to effectively reduce the stochastic noise at the comparable computational cost of SGD, and usually demonstrates promising performance in various applications.

[1] The stochastic noise and variance have equivalent meanings in the paper

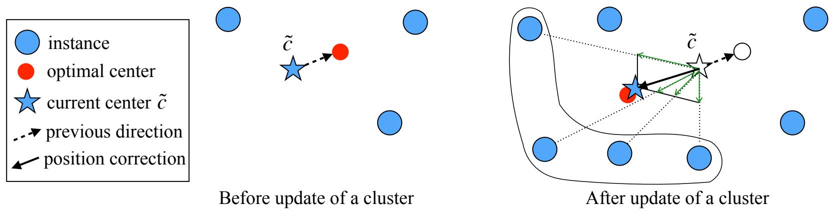


Figure 1: The optimum of the cluster center drifts from the white circle to the red circle. In our method, the current cluster center is corrected by using the gradients corresponding to the newly-joined instances.

One can apply SVRG into k-means to extend its scalability. However, we observe that the objective function of k-means is sharply divergent at iterations when applying the SVRG directly. The reason is that the optimization objective of k-means is jointly dominated by the parameter (cluster centers) and instance partitions (clusters). In specific, the cost function and the expectation of its gradient corresponding to each cluster are changed with the variation of instance partitions. Directly applying SVRG to k-means will first search an optimal direction in expectation based on the current partition. After that, the objective function is decreased along the direction. However, when the instance partition changes, this direction may not be optimal or even not be a decreased one. It is called the drift of the cluster centers. As illustrated in Figure 1, the optimal cluster center drifts when some instances join in the cluster at the iteration. The direction based on the previous instance partition makes the cluster center far from the current optimum. Although it is indeed the optimal direction corresponding to the previous instance partition, it leads to the increase of the objective function for the current instance partition. According to the above discussion, we argue that it is not trivial to apply SVRG into k-means.

To make SVRG applicable in k-means, we develop a simple while effective mechanism to solve the drift of the optimal parameters in this paper. The basic idea is to correct the position of the cluster centers when the corresponding instance partitions are changed. In specific, the cluster centers are corrected by using the gradients corresponding to those newly-joined or newly-left instances at every iteration. This idea can be illustrated from Figure 1. The cluster center is corrected by using the gradients corresponding to the newly-joined instances. The proposed mechanism enables the resultant algorithms to adopt a constant learning rate to update the parameters, which guarantees its efficient convergence. After that, we instantiate two specific algorithms, termed as VRKM and VRKM++, where VRKM samples an instance randomly and updates a cluster center by using a variance reduced gradient at every iteration. VRKM++ is more efficient than VRKM because it decreases much computational cost of the variance reduced gradient. Comprehensive experiments have been conducted on four large-scale datasets. As shown, our algorithms demonstrate significant improvements on the efficiency and clustering performance. Especially, VRKM++ achieves more than $4\times$ speedup on the datasets Oxford and Pittsburgh which contain $12,534,635$ and $7,979,479$ instances, respectively.

The organization of the paper has been presented as follows. Section 2 summaries the related work. Section 3 presents the preliminaries. Section 4 presents the design of the variance reduced k-means. Section 5 presents the optimization of the variance reduced k-means. Section 6 presents the extensive empirical studies. Section 7 concludes the paper.

2. Related work

Lloyd proposes the batch k-means [8]. Since the classic k-means algorithm is slow for large datasets, Bottou and Bengio have proposed a SGD variant of k-means which updates the cluster centers by sampling an instance randomly at every iteration [5]. The SGD variant of k-means is efficient to be performed, but its solution is a victim of the stochastic noise. We present the reason from an optimization view. Generally, an optimization problem is usually formulated as

$$\min_{\omega} f(\omega) = \frac{1}{n}\sum_{i=1}^{n} f_i(\omega) \tag{1}$$

where $f(\omega)$ is usually denoted by an objective function, training loss, loss function or cost function etc equivalently. It is a finite-sum optimization objective consisting of n functions, namely $f_i(\omega)$ with $i \in \{1, 2, ..., n\}$. n is the number of instances in the training dataset, and ω is the parameter we need to train. The batch gradient descent is to update ω by using a batch gradient: ∇f, that is:

$$\omega_{t+1} = \omega_t - \eta \nabla f(\omega_t) = \omega_t - \eta \left(\frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\omega_t) \right). \tag{2}$$

Here, t represents the t-th iteration. η represents a learning rate. When n is extremely large, the computational cost of the batch gradient is very high. Instead of scanning the entire training dataset to obtain a batch gradient, SGD updates ω by sampling an instance, e.g. x_{i_t} with $i_t \in \{1, 2, ..., n\}$ randomly. Here, i_t represents the index of the picked instance at the t-th iteration. Thus, we update the parameter as:

$$\omega_{t+1} = \omega_t - \eta \nabla f_{i_t}(\omega_t). \tag{3}$$

SGD is efficient to reduce the objective function because of

$$\mathbb{E} \nabla f_{i_t}(\omega_t) = \mathbb{E} \left(\frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\omega_t) \right) = \nabla f(\omega_t) \tag{4}$$

where $\mathbb{E}$ represents the expectation operator with respect to i_t. However, the stochastic gradient $\nabla f_{i_t}(\omega_t)$ is usually not equivalent to the batch gradient $\nabla f(\omega_t)$, which is denoted by the variance or stochastic noise. Therefore, SGD usually leads to a low-quality solution due to the stochastic noise. Generally, a decaying learning rate, e.g. $\eta = \frac{1}{t}\eta_0$, is used to reduce the noise where η_0 is a constant. However, it impairs the convergence rate of SGD with the increase of the iterations. Sculley proposes a mini-batch variant of k-means, which reduces the stochastic noise effectively [6], but is still a victim of the stochastic noise.

K-means is performed at two phases alternatively. The first phase needs to compute the nearest cluster center for every instance. The second phase needs to update the cluster center. There are some excellent work focusing on accelerating k-means at the first phrase [9, 10]. For example, the triangle inequality and distance bounds are usually used to reduce the computational cost at the first phase [9, 10]. Additionally, some other impressive studies present exciting results on accelerating k-means. For instance, the binary code learning is used to decrease the memory consumption, and thus the cost of the distance computation is decreased significantly by using Hamming metric [11]. Those great researches are orthogonal to our methods, and can be embedded into our methods to yield more efficiency development. In the paper, we focus on accelerating k-means via variance reduced gradients at the second phase. To the best of our knowledge, this paper is the first work to improve k-means by using the variance reduction technique.

The variance reduction technique has been widely used in variants of SGD to improve the convergence performance [12]. It is effective to reduce the

Table 1: The illustration of symbols and their notations.

Symbols	Notations
$X \in \mathbb{R}^{n\times d}$	The data matrix
X_c	The instances whose cluster has a center c
$x \in \mathbb{R}^{1\times d}$	An instance denoted by a $d \times 1$ column vector
$C \in \mathbb{R}^{d\times K}$	The matrix of cluster centers
$c \in \mathbb{R}^{d\times 1}$	A cluster center
c_*	The optimal cluster center
K	The number of clusters
$v[c]$	The number of instances in the cluster c
∇	The gradient operator
∇f_{i_t}	The stochastic gradient at the tth iteration
$\nabla \tilde{f}_{i_t}$	The noise reducer at the tth iteration
$\nabla \tilde{f}$	The batch gradient
$\mathbb{E}$	The expectation operator
γ	The variance reduced gradient
T	The epoch size
η	The learning rate
i_t	The random number at the ith iteration
$\|\cdot\|$	The l_2 norm of a vector defaultly

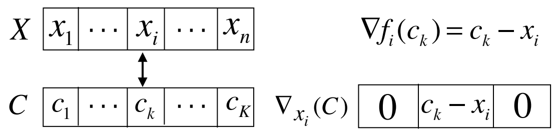


Figure 2: The illustration of the basic data structure and operations where the cluster center c_k is nearest to x_i.

stochastic noise caused by the stochastic gradient at every iteration. Since their computational cost is comparable to SGD, they can be performed efficiently. Those variants of SGD include SVRG [7], SAGA [13], SAG [14], S2GD [15], EMGD [16], SVRG++ [17], AdaSVRG [18] etc. In the paper, we pick a popular variant: SVRG to improve k-means due to its simpleness. Other advanced variance reduction techniques may be used to further the acceleration of k-means, which is left as the future work.

3. Preliminaries

In this section, the symbols and their notations are presented in Table 1. The basic data structure and definitions are then presented. After that, we present k-means from the optimization view. Finally, we present the variance reduction technique used in SVRG.

3.1. Data structure and basic definitions

As illustrated in Figure 2, the training dataset X is a $d \times n$ matrix where n represents the number of instances, and d is the dimension of an instance. The set of instances which belongs to a cluster c is denoted by X_c. x_i with $i \in \{1, 2, ..., n\}$ represents an instance, which is a d-dimensional column vector. c is a d-dimensional column vector, which represents the center of X_c. $\nabla f(c)$ represents the batch gradient with respect to c. $\nabla f_i(c)$ represents the stochastic gradient corresponding to the instance x_i. K is the number of clusters. We use a $d \times K$ matrix C to represent all the K centers such that $C = [c_1, c_2, ..., c_K]$.

Define 1 (**Cluster membership for an instance**). *Given a cluster center set $C = [c_1, ..., c_K]$, the nearest cluster center of an instance x is denoted by*

Algorithm 1 The previous variant of k-means

```
Require: The number of clusters $K$. The dataset $X$.
 1: Initialize every cluster center $c$ with $c \in C$ by picking an instance $x$ from $X$ randomly;
 2: The cluster size $v[c]$ with $c \in C$ is initialized by 0.
 3: for $i = 1$ to $t$ do
 4:     $m$ instances are picked from $X$ randomly, and stored in $M$;
 5:     for $x \in M$ do
 6:         Find the nearest cluster center of $x$, i.e. $x\{C\}$;
 7:     for $x \in M$ do
 8:         $c = x\{C\}$;
 9:         $v[c] = v[c] + 1$;
10:         $\eta = \frac{1}{v[c]}$;                ◊ Decaying learning rate.
11:         $c$=$c$−$\eta(c - x)$;                  ◊ Update rule is like that of SGD.
    return $c$;
```

$x\{C\} \in \mathbb{R}^{d\times 1}$ which is one of the K centers. The index of the center is denoted by $\mathcal{I}_{x\{C\}}$ which is an integer ranging from 1 to K.

Define 2 **(Gradient with respect to $x\{C\}$).** *Given a cluster center set $C = [c_1, ..., c_K]$ and an instance x. The gradient with respect to $x\{C\}$ is denoted by $\nabla_x(C) \in \mathbb{R}^{d\times K}$. The $\mathcal{I}_{x\{C\}}$-th column of $\nabla_x(C)$ is $x\{C\} - x$, and the other columns are zeros.*

$x\{C\}$ represents the nearest center of x. $\nabla_x(C)$ represents the gradient with respect to $x\{C\}$. As shown in Figure 2, if x_i belongs to the cluster c_k, $\nabla_x(C)$ is obtained by using the k-th column of C to subtract x and setting other columns to be 0. There may be a little confusion between C and ω in the paper. When we discuss a general optimization problem, its parameter is denoted by ω. When we focus on the optimization objective function of k-means, its parameter is denoted by C.

3.2. *Optimization view of k-means*

The objective function of k-means clustering is:

$$\min_{C=\{c_1,c_2,...,c_K\}} \frac{1}{2}\sum_{i=1}^{K}\sum_{x\in X_{c_i}} \| c_i - x \|_2^2 \, . \tag{5}$$

Here, K is the number of the clusters, and c_i is the nearest cluster center of x. Both the SGD and mini-batch variants of k-means can be thought as optimization methods to solve (5) [4]. As illustrated in Algorithm 1, when $m = 1$ holds, Algorithm 1 is the SGD variant of k-means. When $m > 1$ and $m < n$ hold, Algorithm 1 is the mini-batch k-means [6]. Additionally, Line 10 shows that a decaying learning rate is used. The update rule in Line 11 is performed like SGD where $c - x$ is the stochastic gradient corresponding to x. The learning rate η is decayed with the increase of the iterations to reduce the stochastic noise. Considering a large number of the iterations, it is difficult to make a further progress when η is extremely small. Since the stochastic gradient does not equal to the batch gradient, both the SGD and mini-batch k-means are the victims of the stochastic noise, thus leading to low-quality solutions.

3.3. *SVRG: SGD with the variance reduction technique*

As illustrated in (1) and (3), a general optimization problem can be solved by using SGD. The stochastic gradient in SGD does not usually equal to the batch gradient, which leads to the stochastic noise. SVRG is effective to reduce the stochastic noise. The key idea is to use a variance reduced gradient to update the parameter. The variance reduced gradient is

$$\gamma=\nabla f_{i_t}-\nabla\tilde{f}_{i_t}+\nabla\tilde{f} \tag{6}$$

where i_t with $i_t \in \{1, 2, ..., n\}$ is picked randomly. ∇f_{i_t} is a stochastic gradient, $\nabla\tilde{f}_{i_t}$ is a noise reducer, and $\nabla\tilde{f}$ is a snapshot of the batch gradient. SVRG keeps a good property by designing $\nabla\tilde{f}_{i_t}$ and $\nabla\tilde{f}$ appropriately, that is:

$$\lim_{t\to\infty} \mathbb{E}\gamma = \lim_{t\to\infty} \Big(\mathbb{E}(\nabla f_{i_t} - \nabla\tilde{f}_{i_t} + \nabla\tilde{f})\Big) = 0 \tag{7}$$

where ∇f is the batch gradient with respect to the current parameter. That is, γ is converged to 0 with the increase of the iterations.

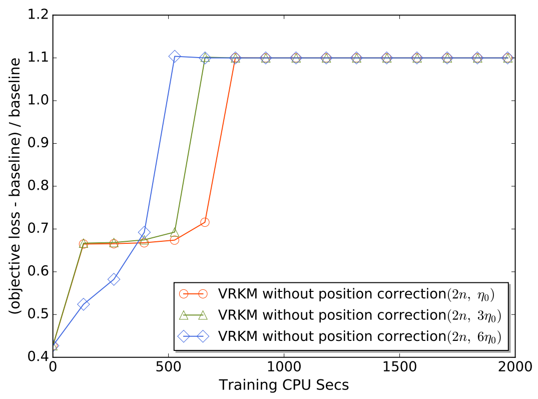


Figure 3: Variance Reduced K-Means (VRKM) is divergent on the dataset: CIFAR-100 if SVRG is used trivially.

4. Variance reduced k-means clustering

In this section, we provide a method to solve the position drift problem, and propose a variance reduced k-means named VRKM.

4.1. Position correction

Although SVRG is effective to decrease the stochastic noise, it is challenging to combine it with k-means. As illustrated in Figure 3, if SVRG is used in k-means directly, the objective function is increased, and thus leads to be divergent. The specific meanings of symbols and the metric in Figure 3 are illustrated in the empirical studies.

The reason is that a cluster may be changed between iterations, and the instances within a cluster are not fixed. During the iterations, some new instances may be added into the cluster, and some instances may be removed from the cluster. The batch gradient is jointly dominated by the cluster center and the partition. It is different from (1) because that the component functions in (1) are fixed. The batch gradient in (1) is completely dominated by its parameter ω. This difference leads to the drift of the optimal parameter. As illustrated in Figure 1, the cluster center is updated between iterations. It is easy to see that some new instances are added into the cluster center after an iteration. The

Algorithm 2 Position correction for the cluster center $\tilde{c}$

```
Require: The cluster $X_{\tilde{c}}$ and its size $v$.
1: $\nabla \tilde{f}(\tilde{c}) = \frac{1}{v} \sum_{x \in X_{\tilde{c}}} (\tilde{c} - x)$;    ◊ Batch gradient for $\tilde{c}$.
2: $\tilde{c}^{\text{new}} \leftarrow \tilde{c} - \nabla \tilde{f}(\tilde{c})$;    ◊ $\tilde{c}$ is corrected.
3: return $\tilde{c}$;
```

cluster center thus becomes far from the optimum after the drift of the position. If the cluster is updated, the drift of the cluster center always exists, which makes it impossible to update the center until convergence.

We conduct *position correction* for every cluster center. The position correction is shown in Figure 1. When the cluster center $\tilde{c}$ drifts, it is corrected by the gradients corresponding to those newly-joined or newly-left instances. First, when some instances are added into the cluster $X_{\tilde{c}}$, $\tilde{c}$ is corrected to be close to those newly-joint instances. If some instances are removed from $X_{\tilde{c}}$, $\tilde{c}$ is corrected to be far from those newly-left instances. Second, the gradients corresponding to other instances are used to avoid aggressive correction. In other words, all the gradients corresponding to the instances are involved to correct the position of $\tilde{c}$. The position correction guarantees that the cluster center $\tilde{c}$ is close to the optimum, even though it drifts due to the change of the instance partition. As illustrated in Algorithm 2, the current center $\tilde{c}$ is corrected by using all the gradients corresponding to the instances in the cluster $X_{\tilde{c}}$.

We then update the cluster centers by using the variance reduced gradient iteratively. The variance reduced k-means denoted by VRKM has been illustrated in Algorithm 3. The outer loop (Lines $2-17$) represents that VRKM is organized by epochs. Line 3 requires to scan the entire dataset in order to obtain the nearest cluster center for every instance. Meanwhile, the number of instances within a cluster has been obtained. Line 4 is the *position correction* which solves the position drift of the optimal cluster center according to Algorithm 2. The inner *for* loop (Lines $9-16$) means to update the cluster centers by using the variance reduced gradient γ_t. In specific, Line 13 represents the variance reduced gradient. Line 14 shows that the constant learning rate is used to update the

Algorithm 3 VRKM: variance reduced k-means

```
Require: The number of clusters $K$. The dataset $X$. The constant learning rate $\eta$. The epoch size $T$.
 1: Initialize every $c \in \tilde{C}$ with an instance picked from $X$ randomly;
 2: repeat
 3:     Update the nearest cluster center for every instance $x_i$ with $i \in \{1, 2, ..., n\}$ according to $\tilde{C}$, and thus obtain the instance partitions $\{X_{c_1}, ..., X_{c_K}\}$;
 4:     Conduct position correction for a cluster center $c_i$ with $1 \le i \le K$, and obtain $\tilde{c}_i^{\text{new}}$ according to Algorithm 2;
 5:     $C_0 = \tilde{C}^{\text{new}} = \{\tilde{c}_1^{\text{new}}, ..., \tilde{c}_K^{\text{new}}\}$, and let $X_{\tilde{c}_i^{\text{new}}} = X_{c_i}$ for $1 \le i \le K$;
 6:     Obtain $x_i\{\tilde{C}^{\text{new}}\}$, $\mathcal{I}_{x_i\{\tilde{C}^{\text{new}}\}}$ and $\nabla_{x_i}(\tilde{C}^{\text{new}})$ for every instance $x_i$ with $1 \le i \le n$ based on the instance partition $\{X_{\tilde{c}_1^{\text{new}}}, ..., X_{\tilde{c}_K^{\text{new}}}\}$;
 7:     $\nabla\tilde{f}(\tilde{c}_i^{\text{new}}) = \frac{1}{v_i} \sum_{x \in X_{\tilde{c}_i^{\text{new}}}} (\tilde{c}_i^{\text{new}} - x)$ for $1 \le i \le K$;
 8:     $\nabla\tilde{f}(\tilde{C}^{\text{new}}) = \{\nabla\tilde{f}(\tilde{c}_1^{\text{new}}), \nabla\tilde{f}(\tilde{c}_2^{\text{new}}), ..., \nabla\tilde{f}(\tilde{c}_K^{\text{new}})\}$;
 9:     for $t = 0, 1, ..., T-1$ do
10:         Pick an index $i_t$ from $\{1, 2, ..., n\}$ randomly;
11:         Find the nearest cluster center from $C_t$ for $x_{i_t}$, and thus obtain $x_{i_t}\{C_t\}$, $\mathcal{I}_{x_{i_t}\{C_t\}}$ and $\nabla_{x_{i_t}}(C_t)$;
12:         if $x_{i_t}\{C_t\} \ne x_{i_t}\{\tilde{C}^{\text{new}}\}$ or $\mathcal{I}_{x_{i_t}\{C_t\}} \ne \mathcal{I}_{x_{i_t}\{\tilde{C}^{\text{new}}\}}$ then
13:             $\gamma_t = \nabla_{x_{i_t}}(C_t) - \nabla_{x_{i_t}}(\tilde{C}^{\text{new}}) + \nabla\tilde{f}(\tilde{C}^{\text{new}})$;
14:             $C_{t+1} = C_t - \eta\gamma_t$;
15:         else   $C_{t+1} = C_t$;
16:     $\tilde{C} = C_T$;
17: until convergence;
18: return $\tilde{C}$;
```

parameter.

It is worth noting that $\nabla_{x_{i_t}}(C_t)$ is obtained based on X_{C_t} which may be updated at every iteration. But, $\nabla_{x_{i_t}}(\tilde{C}^{\text{new}})$ and $\nabla\tilde{f}(\tilde{C}^{\text{new}})$ are obtained based on the $X_{\tilde{C}^{\text{new}}}$ which is constant during iterations in an epoch. Thus, although

$C_0 = \tilde{C}^{\text{new}}$ at $t = 0$, the stochastic gradients $\nabla_{x_{i_t}}(C_t)$ and $\nabla_{x_{i_t}}(\tilde{C}^{\text{new}})$ are different if x_{i_t} belongs to a different cluster against the partition $X_{\tilde{C}^{\text{new}}}$.

Compared with previous k-means, i.e. Algorithm 1, VRKM is organized as epochs, and the cluster centers are updated iteratively during an epoch. The variance reduced gradient, i.e. γ_t is equivalent to the batch gradient in expectation. Thus, the stochastic noise has been decreased effectively. When we pick an instance x_{i_t} randomly, the stochastic gradient $\nabla_{x_{i_t}}(C_t)$ and the noise reducer $\nabla_{x_{i_t}}(\tilde{C})$ are computed at every iteration. The computational cost is comparable to the SGD variant of k-means. VRKM can be performed more efficiently. According to Algorithm 2, the position correction needs to compute a stochastic gradient for every instance. If those stochastic gradients are saved in memory, the noise reducer $\nabla_{x_{i_t}}(\tilde{C})$ does not need to be computed during iteratively update of the parameter. Thus, the computational cost every iteration is decreased at the cost of memory significantly.

4.2. The constant learning rate

Benefiting from the position correction mechanism, the variance reduced k-means is accelerated by using a constant learning rate. The variance reduced gradient equals to the batch gradient in expectation according to (7). As with the increase of iterations, the cluster center c_t at the current iteration is close to the optimum denoted by c_*. Since

$$\begin{aligned} & \eta \lim_{t\to\infty} \mathbb{E}\gamma_t \\ = \; & \lim_{t\to\infty} \mathbb{E}C_{t+1} - \lim_{t\to\infty} \mathbb{E}C_t \\ = \; & C_* - C_* = \mathbf{0}. \end{aligned}$$

That is to say, $\lim_{t\to\infty} (\mathbb{E}\gamma_t) = 0$ holds when the learning rate η is a constant [7]. Benefiting from this property, the variance reduced k-means can be accelerated by using a constant learning rate, which has a significant advantage over the pervious methods. No matter the SGD variant of k-means or the mini-batch k-means, a decaying learning rate is used to decrease the stochastic noise. The decaying learning rate makes k-means converge very slowly after a number of iterations. It is difficult to make a further progress with the increase of the iteration. This weakness of the previous methods has been overcome by using a constant learning rate in the variance reduced k-means. Numerical results have verified the advantage of the constant learning rate, which is efficient to converge the objective function of k-means.

5. Algorithm optimization

In this section, a new variant of the variance reduced k-means named VRKM++ is proposed, which decreases much computational cost of VRKM.

5.1. VRKM++: VRKM without batch gradients

VRKM has to compute the batch gradient twice in an epoch. The first batch gradient is computed for correcting the position of the cluster center (see Line 8 in Algorithm 3), and the second is used in the variance reduced gradient (see Line 13 in Algorithm 3). If the batch gradient is stored in memory, the batch gradient needs to be computed only once, but it consumes much memory for a large dataset. Generally, for a large dataset, both batch gradients need to be computed by scanning every instance of the cluster, which increases the computational cost. The computation of the batch gradient needs to scan the entire dataset, which is time-consuming. VRKM++ is proposed to optimize VRKM, which does not lead to the computational cost of the batch gradient.

VRKM++ decreases the computational cost by the following observation:

$$\tilde{c}^{\text{new}} \leftarrow \tilde{c} - \nabla\tilde{f}(\tilde{c}) = \tilde{c} - \left(\tilde{c} - \frac{1}{v_k}\sum_{x\in X_c} x\right) = \frac{1}{v_k}\sum_{x\in X_{\tilde{c}}} x = \bar{X}_{\tilde{c}} \tag{8}$$

where $\tilde{c}$ is corrected via the position correction. That is, the current cluster center is corrected by the average of the instances in this cluster. In fact, we can conduct the position correction by scanning the cluster only once, and we do not have to compute the batch gradient $\nabla\tilde{f}(\tilde{c})$. First, we obtain the new center $\tilde{c}$ with $\tilde{c} = \bar{X}_{\tilde{c}}$ by scanning the cluster according to (8). Second, we compute the

Algorithm 4 VRKM++: variance reduced k-means without batch gradients

```
Require: The number of clusters $K$. The dataset $X$. The constant learning rate $\eta$. The epoch size $T$.
1: Initialize each $c \in \tilde{C}$ with instances picked from $X$ randomly;
2: repeat
3:     Update the nearest cluster center for every instance $x_i$ with $i \in \{1, 2, ..., n\}$ according to $\tilde{C}$, and thus obtain the instance partitions $\{X_{c_1}, ..., X_{c_K}\}$;
4:     $C_0 = \tilde{C}^{\text{new}} = (\bar{X}_1, \bar{X}_2, ..., \bar{X}_K)$, and let $X_{\tilde{c}_i^{\text{new}}} = X_{c_i}$ for $1 \le i \le K$; ◊ Position correction.
5:     Obtain $x_i\{\tilde{C}^{\text{new}}\}$, $\mathcal{I}_{x_i\{\tilde{C}^{\text{new}}\}}$ and $\nabla_{x_i}(\tilde{C}^{\text{new}})$ for every instance $x_i$ with $1 \le i \le n$ based on the instance partition $\{X_{\tilde{c}_1^{\text{new}}}, ..., X_{\tilde{c}_K^{\text{new}}}\}$;
6:     for $t = 0, 1, ..., T-1$ do
7:         Pick an index $i_t$ from $\{1, 2, ..., n\}$ randomly;
8:         Find the nearest cluster center from $C_t$ for $x_{i_t}$, and thus obtain $x_{i_t}\{C_t\}$, $\mathcal{I}_{x_{i_t}\{C_t\}}$ and $\nabla_{x_{i_t}}(C_t)$;
9:         if $x_{i_t}\{C_t\} \neq x_{i_t}\{\tilde{C}^{\text{new}}\}$ or $\mathcal{I}_{x_{i_t}\{C_t\}} \neq \mathcal{I}_{x_{i_t}\{\tilde{C}^{\text{new}}\}}$ then
10:            $\gamma_t = \nabla_{x_{i_t}}(C_t) - \nabla_{x_{i_t}}(\tilde{C}^{\text{new}})$;
11:            $C_{t+1} = C_t - \eta\gamma_t$;
12:        else $C_{t+1} = C_t$;
13:    $\tilde{C} = C_T$;
14: until convergence;
15: return $\tilde{C}$;
```

batch gradient immediately, that is,

$$\nabla \tilde{f}(\tilde{c}_i^{\text{new}}) = \tilde{c}_i^{\text{new}} - \tfrac{1}{v_i} \sum_{x \in X_{\tilde{c}_i}^{\text{new}}} x = \mathbf{0}. \tag{9}$$

Based on this observation, the batch gradient, namely $\nabla \tilde{f}(\tilde{C}^{\text{new}})$ does not have to be computed, thus decreasing much computational cost of VRKM. As illustrated in Algorithm 4, line 4 represents the position correction where all the cluster centers are corrected by the average of the instances within a cluster. Line 10 means that the variance reduced gradient does not need the computation of the batch gradient. Benefiting from the optimization, VRKM++ is performed more efficient than VRKM.

Note that Line 4 in Algorithm 4 is the average of the instance within a cluster. As we know, the classic k-mean also uses the average of instance to update the cluster centers for the next iteration. However, the average strategy in VRKM++ has the different goal from the classic k-means. The average strategy is used to obtain the initial cluster centers, i.e. $\tilde{C}^{\text{new}}$. The new initial cluster centers are yielded to update the cluster centers via the variance reduced gradients, and will be discarded at the end of the iteration. The new-learned cluster center C_T will be used to update the cluster membership of instances at the next iteration, which is different from the classic k-means.

To make it clear, we illustrate our variance reduced k-means according to Figure 4. As shown in the top-left subfigure, the instances are partitioned into two clusters. x belongs to the blue cluster and its nearest cluster center is c_2, that is, $x\{\tilde{C}\} = c_2$. After the position correction, x's nearest cluster center is c_1, namely $x\{C_0\} = c_1$. Thus, the cluster centers are updated via the variance reduced gradients. As shown in the top-right subfigure, $\nabla_x(C_0) = c_1 - x$ and $\nabla_x(\tilde{C}) = c_2 - x$ hold. Then, c_1 and c_2 are updated in the bottom-left subfigure. Finally, the instances are partitioned into new clusters as illustrated in the bottom-right subfigure.

5.2. Parameter settings

Both VRKM and VRKM++ are organized as epochs and the entire dataset needs to be scanned to update the cluster in an epoch, which is time-consuming. A large epoch size T helps to make the objective function converge efficiently. The epoch size T is usually set to be comparable to the size of the training data, e.g. $0.25n$, n, or $2n$. Alternatively, the epoch size T can be set by using a heuristic method. For example, T can be increased exponentially e.g. $T = 2^s$ where s represents the s-th epoch.

The learning rate η is a constant, and needs to be set before the start of VRKM. It plays an important role in the performance of our methods. A relatively large

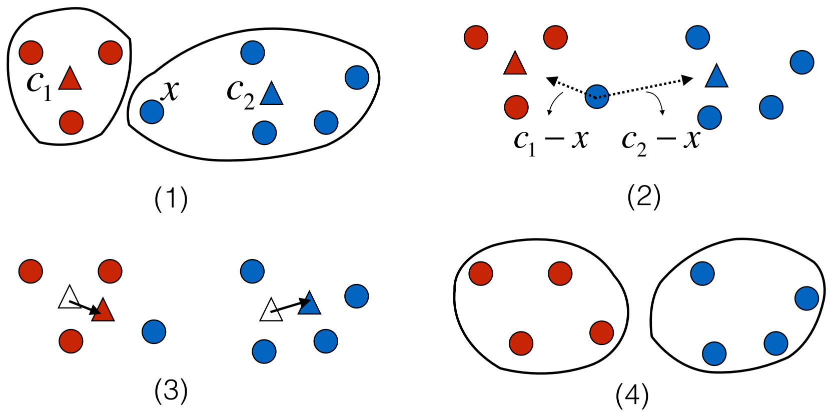


Figure 4: Cluster centers are updated via variance reduced gradients.

η helps to improve the convergence performance of VRKM. However, VRKM will be divergent when η is extremely large. Empirically, the constant learning rate can be set as $\eta = \frac{K}{n}$. Here, K is the number of clusters, and n is the numebr of instances. Note that this setting of the learning rate is conservative. It can be set larger than $\frac{K}{n}$ for a large dataset to obtain the improvement of the convergence performance. Empirical studies show that VRKM and VRKM++ can be performed efficiently by using the heuristic method, and outperform their counterparts significantly.

Additionally, there are some impressive researches investigating the settings of those hyper-parameters when using the variance reduction techniques [19, 20]. Although we do not use those advanced methods in VRKM and VRKM++, it is not difficult to use those methods to set the hyper-parameters including the epoch size and the learning rate. For example, the epoch size is set heuristically in [19]. An adaptive method to set an optimal learning rate is proposed in [20]. The researches can be used to improve VRKM and VRKM++, and we recommend readers to read the papers for more details.

6. Empirical studies

In this section, we conduct extensive empirical studies on a Red Hat Enterprise 64-bit Linux workstation with 12-core Intel Xeon Westmere EP CPU 2.93 GHz and 48 GB memory. Both VRKM and VRKM++ are compared with the batch k-means denoted by *batch KM* [8], the SGD k-means denoted by *SGD-KM* [5], the mini-batch k-means denoted by *mini-batch KM* [6], and the state-of-the-art algorithm denoted by *growbatch-rho* [9]. As far as we know, *growbatch-rho* is the newest variant of k-means which is related to our methods.

K-means is very sensitive to the initial conditions. The initial seeds of k-means usually have significant impacts on the clustering result. Before conducting the experiments, we pick the initial seeds randomly. After that, the seeds are fixed, and we use them to conduct clustering for all algorithms. It is essential to remove the impact of the randomness due to the seeds, and to conduct the comparison for all algorithms fairly. Additionally, as we have illustrated in Section 3.2, the previous methods including *mini-batch KM* and *SGD-KM* use the decaying learning rate. In other words, their learning rates are shrunk with the increase of the iteration. The learning rate of *batch KM* is the reciprocal of the cluster size. We do not change the settings of those methods. The source code of the previous methods, namely *batch KM*, *mini-batch KM* and *SGD-KM* are public, and we use the open source code to conduct the evaluation studies [2]. The implementation of *growbatch-rho* is public, and we use the open source code to conduct the evaluation studies [3].

6.1. Datasets

The comparisons are conducted on four datasets: Oxford, Pittsburgh, CIFAR-100, and Caltech-256. The details of those datasets are illustrated in Table 2.

- Oxford[4]: It consists of 5062 images collected from Flickr by searching for

[2] http://code.google.com/p/sofia-ml
[3] https://github.com/idiap/eakmeans
[4] http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/

Table 2: Statistics of four large-scale datasets.

Datasets	#Instances	#Dimension	#clusters
Oxford	$12,534,635$	128	1000
Pittsbour	$7,979,479$	128	2000
CIFAR-100	$60,000$	3072	100
Caltech-256	$30,607$	4096	257

particular Oxford landmarks [21]. Every image is used to generate about $2000-3000$ SIFT descriptors by running the open source tool[5] [22]. We finally use these SIFT descriptors as the instances to perform k-means clustering.

- Pittsbour[6]: It is a geotagged image database is formed by $254,064$ perspective images. Those perspective images are generated from $10,586$ Google Street View panoramas of the Pittsburgh area [23]. We generate about $2000-3000$ SIFT descriptors for every image by using the open source tool[7] [22], and finally use these SIFT descriptors as the instances to conduct k-means clustering.

- CIFAR-100[8]: It contains $60,000$ 32×32 pixels images [24]. Each image is represented by a 3072-dimensional column vector which is generated by using the pixels of the raw image. We conduct k-means clustering on those vectors.

- Caltech-256[9]: It consists of $30,607$ images [25]. We use a 4096-dimensional feature to represent every image. Each feature is extracted by the convolution neural networks model [26]. Those features are used to perform k-means clustering.

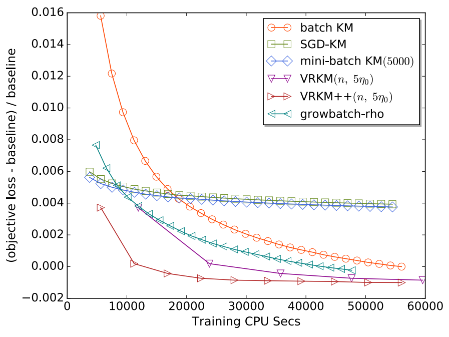
[5] http://cmp.felk.cvut.cz/~perdom1/hesaff/

[6] http://www.ok.ctrl.titech.ac.jp/~torii/project/repttile/

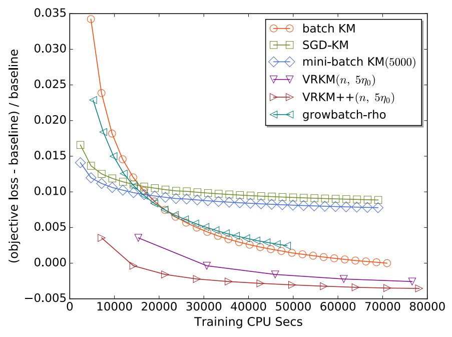
[7] http://cmp.felk.cvut.cz/ perdom1/hesaff/

[8] http://www.cs.toronto.edu/~kriz/cifar.html

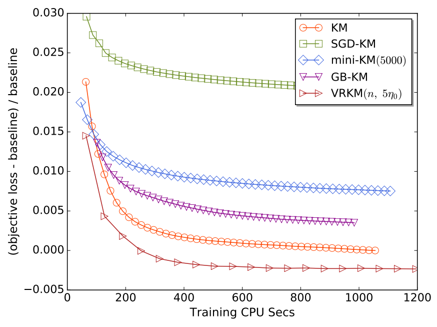
[9] http://www.vision.caltech.edu/Image_Datasets/Caltech256/



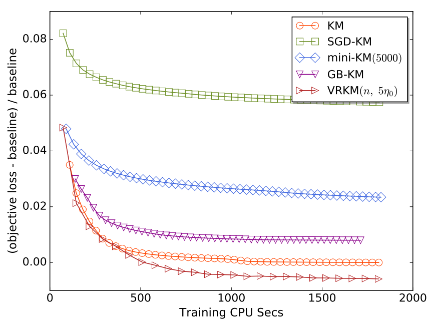
(a) Oxford



(b) Pittsburgh



(c) CIFAR-100



(d) Caltech-256

Figure 5: The comparison of the objective loss on four large datasets.

6.2. Convergence performance

Objective loss: We conduct the comparison of the objective loss on four datasets. The x-axis represents CPU seconds during the training of the parameter. The y-axis represents the decrease of the objective function against a baseline. The baseline is obtained by running *batch KM* for the given time.

As illustrated in Figure 5, both VRKM and VRKM++ have advantages on decreasing the objective loss for the given time. The advantage is more significant on large datasets, which means that VRKM and VRKM++ are efficient to conduct large-scale k-means clustering tasks. The main reason is that both VRKM and VRKM++ update the cluster centers by using variance reduced gradients, which is equivalent to *batch KM* in expectation. Benefiting

Table 3: The comparison of time consumption and the final objective loss on four datasets.

Datasets	Oxford/Pittsburgh		
	Time	Speedup	Loss
batch KM	16.11/20h	$1.00\times/1.00\times$	$6,574,972/3,820,667$
mini-batch KM	15.56/19.31h	-/-	$6,599,589/3,850,275$
SGD-KM	15.56/19.31h	-/-	$6,600,734/3,854,394$
growbatch-rho	12.78/13.50h	$1.26\times$/-	$6,572,795/3,830,066$
VRKM	**7.78**h/**7.78**h	**2.07**×/**2.57**×	$\mathbf{6,569,367/3,810,825}$
VRKM++	**3.89**h/**3.61**h	**4.14**×/**5.54**×	$\mathbf{6,568,354/3,810,825}$

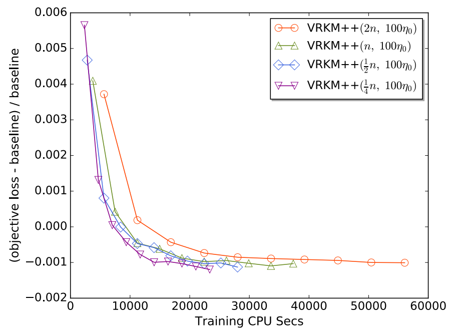
Datasets	CIFAR-100/Caltech-256		
	Time	Speedup	Loss
batch KM	0.3/0.5h	$1.00\times/1.00\times$	$6,147,102/110,830,416$
mini-batch KM	0.31/0.5h	-/-	$6,193,337/117,204,216$
SGD-KM	0.31/0.5h	-/-	$6,271,261/113,424,464$
growbatch-rho	0.25/0.476h	-/-	$6,161,949/111,715,468$
VRKM	**0.147**h/**0.25**h	**2.04**×/**2.00**×	$\mathbf{6,134,762/110,337,072}$
VRKM++	**0.069**h/**0.139**h	**4.30**×/**3.60**×	$\mathbf{6,132,739/110,177,936}$

from the lower computational cost, both VRKM and VRKM++ complete more iterations than *batch KM* for the given time, and thus yield much more decrease of the objective loss than *batch KM*. It is noting that if we run *batch KM* for enough long time, it will decrease the objective loss equivalently or even more than our methods. However, this leads to much time consumption, which is not bearable. For example, when we run *batch KM* on Pittsburgh, it takes more than 30 hours to decrease the objective loss like VRKM++ in Figure 5(b). The SGD and mini-batch variants of k-means are victims of the stochastic noise which impedes the decrease of the objective loss. *growbatch-rho* has an advantage to decrease the objective loss on the Oxford dataset. But, it is outperformed by our methods in all evaluations.

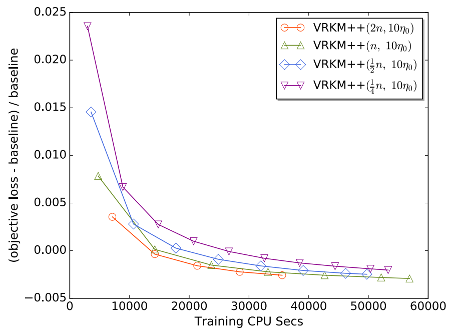
Speedup: As illustrated in Table 3, *Time* is recorded from the start of an algorithm, and is collected by using the unit of hour. In order to compare the time consumption fairly, we run all the algorithms until that their objective loss is decreased to a baseline. The baseline is not easy to be determined because that the convergence rates of all the previous methods vary a lot. Since *batch KM* is the basic variant of k-means, we use its final objective loss as the baseline, and try to collect the time consumption for every algorithm. However, the *SGD-KM*, *mini-batch KM* and *growbatch-rho* sometimes cannot decrease the objective loss to the baseline for the given time. Therefore, we shut them down when we find that they cannot decrease the objective loss to the baseline, and collect their total time consumption. *Speedup* is computed by dividing the time consumption of *batch KM*. If an algorithm cannot decrease the objective loss to the baseline, it has no speedup. Additionally, *Loss* is the final objective loss before those algorithms are shut down.

We can obtain some advantages of our methods from Table 3. First, we can find that both VRKM and VRKM++ obtain significant speedup on the time consumption. Generally, VRKM obtains more than 2$\times$ speedup, and VRKM++ obtains more than 4$\times$ speedup. The speedup becomes more significant on large datasets. As we have shown, the speedup of VRKM and VRKM++ benefits from the variance reduced gradients when we update the cluster centers. Second, both VRKM and VRKM++ reduce the objective loss more effectively than the other methods. This observation verifies the advantage of VRKM and VRKM++ again. The reason is that the computational cost of VRKM and VRKM++ are comparable to the SGD variant of k-means. Meanwhile, SVRG reduces the stochastic noise effectively, and updates the parameters by using a constant learning rate. It is highlighted that VRKM++ performs best on the decrease of the objective loss and the time consumption. The reason is that VRKM++ does not need to compute the batch gradient at every epoch, which decreases much computational cost of VRKM.

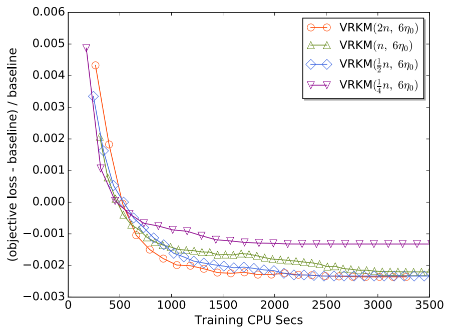
Epoch size: We provide the comparison of the convergence performance by varying the epoch size. As illustrated in Figure 6, we can find that the settings of the epoch size have a slight impact on the convergence performance and the



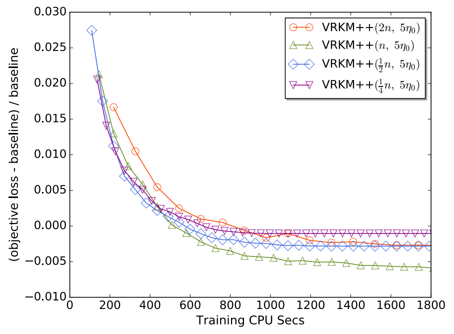
(a) Oxford



(b) Pittsburgh



(c) CIFAR-100

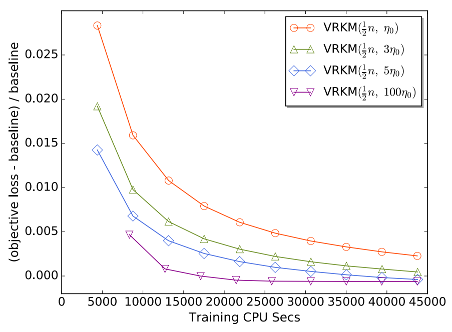


(d) Caltech-256

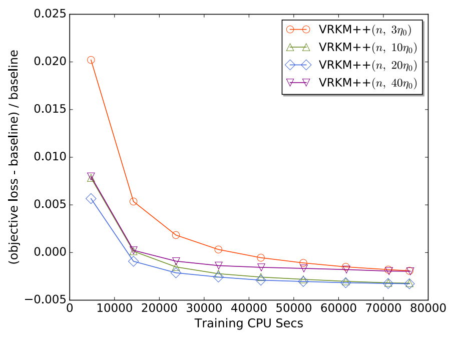
Figure 6: The comparison of the convergence performance on four datasets by varying the epoch size.

decrease of the objective loss. By varying the epoch size from $\frac{1}{4}n$ to $2n$, the performance does not have much improvement, especially for large datasets. Based on the observation, we recommend to set the epoch size to be comparable to the size of the training dataset such as $0.5n$ or n. This setting of the epoch size is good enough to converge the parameter and decrease the objective loss.

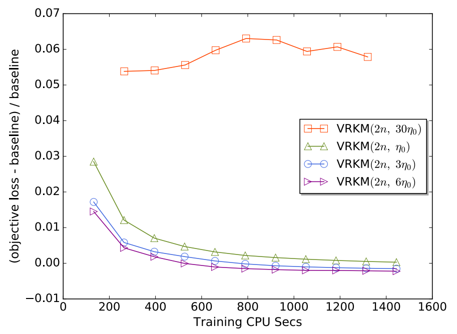
Learning rate: As illustrated in Figure 7, the comparison of the convergence performance is conducted by varying the learning rate. We use $\eta_0 = \frac{K}{n}$ as a baseline, and vary the learning rate by multiplying a positive integer. Here, n and K represents the number of instances and clusters, respectively. Generally, it shows that a larger learning rate improves the convergence performance. However, when the learning rate is very large, the converge performance is impaired, e.g.



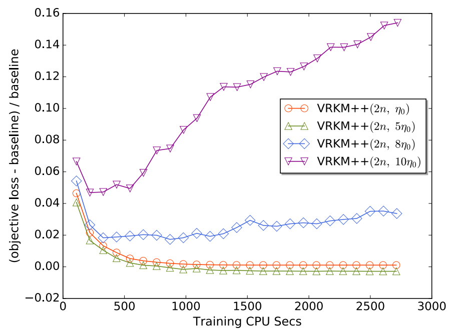
(a) Oxford



(b) Pittsburgh



(c) CIFAR-100



(d) Caltech-256

Figure 7: The comparison of the convergence performance on four datasets by varying the learning rate.

the red line in Figure 7(c) and the blue and purple lines in Figure 7(d). We can observe that the learning rate can be set much larger than the baseline for large datasets. For example, when the learning rate η is larger than $100\eta_0$ on the dataset Oxford, the improvement of the convergence performance is still significant. In fact, although the baseline, i.e. $\eta_0 = \frac{K}{n}$ is conservative for VRKM and VRKM++, they still outperform the other methods on the convergence performance. We recommend this setting of the learning rate when conducting VRKM and VRKM++ for large-scale clustering tasks.

Table 4: The comparison of clustering performance on CIFAR-100 and Caltech-256.

Dataset	CIFAR-100		
	ACC	NMI	Purity
batch KM	0.2148(0.0015)	0.3701(0.0012)	0.2410(0.0006)
mini-batch KM	0.2122(0.0026)	0.3546(0.0024)	0.2267(0.0012)
SGD-KM	0.1968(0.0034)	0.3357(0.0032)	0.2079(0.0015)
growbatch-rho	0.2155(0.0023)	0.3592(0.0020)	0.2303(0.0011)
VRKM	**0.2246(0.0018)**	**0.3742(0.0014)**	**0.2488(0.0009)**
VRKM++	**0.2246(0.0018)**	**0.3742(0.0014)**	**0.2488(0.0009)**
Dataset	Caltech-256		
	ACC	NMI	Purity
batch KM	0.4736(0.0009)	0.6817(0.0013)	0.5489(0.0011)
mini-batch KM	0.4395(0.0012)	0.6608(0.0016)	0.5091(0.0019)
SGD-KM	0.4259(0.0017)	0.6255(0.0022)	0.4625(0.0024)
growbatch-rho	0.4758(0.0015)	0.6750(0.0017)	0.5411(0.0017)
VRKM	**0.4969(0.0011)**	**0.6929(0.0016)**	**0.5686(0.0014)**
VRKM++	**0.4969(0.0011)**	**0.6929(0.0016)**	**0.5686(0.0014)**

6.3. Clustering performance

The clustering performance is compared on CIFAR-100 and Caltech-256. We adopt three metrics, namely: ACC, NMI, and Purity. In experiments, we repeat every algorithm for ten times by using random seeds as the initial cluster centers. Note that the clustering results are yielded by running those algorithms for a given time. For every algorithm, when the time is exhausted, it will be terminated. Then, we begin to collect the results. As illustrated in Table 4, those values out of parenthese are the mean values. Those values in the parenthese represent the variance. Additionally, we extract those features from images, but we do not conduct fine-tuning for those features.

VRKM and VRKM++ have the same clustering performance for the clustering tasks, and both of them have advantages over their counterparts. We can also find that SGD variant of k-means has the worst performance due to the much stochastic noise. Mini-batch k-means performs better than the SGD variant of k-means, but less than the batch k-means. The reason is that the mini-batch k-means updates the parameter by sampling a mini-batch of instances, which reduces part of the stochastic noise. Besides, although the batch variant of k-means uses the full gradients to update the cluster centers, it usually performs fewer iterations than VRKM and VRKM++ for the given time. Therefore, its clustering performance is not better than our proposed methods. By contrast, both VRKM and VRKM++ update their parameters by using the variance reduced gradients, and perform k-means more efficiently. In other words, they finish more iterations than the previous methods for the given time. Thus, they outperforms the previous methods, and obtain the best clustering performance.

7. Conclusion

SVRG is effective to reduce the stochastic noise. However, it is challenging to be used in k-means due to the drift of the cluster centers. In the paper, we propose a position correction mechanism to solve such challenging problem, and use a constant learning rate to update the parameter in k-means. Furthermore, we present two variants of variance reduced k-means: VRKM and VRKM++. VRKM++ does not have to compute the batch gradient at every epoch, thus decreasing much computational cost than VRKM. Extensive empirical studies show that both VRKM and VRKM++ are efficient to conduct large-scale k-means clustering tasks, and outperform the state-of-the-art method significantly.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (Project no. 60970034, 61170287, 61232016, 61672528 and 61671463). We thank for the help provided by Prof. Xinzhong Zhu who is with the college of Mathematics, Physics and Information Engineering, Zhejiang Normal University, Jinhua, China. Additionally, we thank Cixing intelligent manufacturing research institute, Cixing textile automation research institute, Ningbo Cixing corporation limited and Ningbo Cixing robotics company limited because of their financial support.

References

[1] S. Ray, R. H. Turi, Determination of number of clusters in k-means clustering and application in colour image segmentation, in: Proceedings of International Conference on Advances in Pattern Recognition and Digital Techniques, 1999, pp. 137–143.

[2] S. Chawla, A. Gionis, k-means–: A unified approach to clustering and outlier detection, in: Proceedings of SIAM International Conference on Data Mining, SIAM, 2013, pp. 189–197.

[3] P. Pantel, D. Lin, Discovering word senses from text, in: Proceedings of ACM International Conference on Knowledge Discovery and data mining, 2002, pp. 613–619.

[4] S. Shalev-Shwartz, S. Ben-David, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, 2014.

[5] L. Bottou, Y. Bengio, et al., Convergence properties of the k-means algorithms, in: Proceedings of Advances in Neural Information Processing Systems, 1995, pp. 585–592.

[6] D. Sculley, Web-scale k-means clustering, in: International Conference on World Wide Web, Raleigh, USA, 2010, pp. 1177–1178.

[7] R. Johnson, T. Zhang, Accelerating stochastic gradient descent using predictive variance reduction, in: Proceedings of Advances in Neural Information Processing Systems, Lake Tahoe, USA, 2013, pp. 315–323.

[8] S. Lloyd, Least squares quantization in pcm, IEEE Transactions on Information Theory 28 (2) (1982) 129–137.

[9] J. Newling, F. Fleuret, Nested Mini-Batch K-Means, in: Proceedings of Advances in Neural Information Processing Systems, 2016.

[10] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, T. Mytkowicz, M. Musuvathi, Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup, in: Proceedings of International Conference on Machine Learning, 2015, p. 579–587.

[11] X. Shen, W. Liu, I. Tsang, F. Shen, Q.-S. Sun, Compressed k-means for large-scale clustering, in: Proceedings of AAAI Conference on Artificial Intelligence, 2017, pp. 895–903.

[12] L. Bottou, F. E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, arXiv preprint arXiv:1606.04838.

[13] A. Defazio, F. Bach, S. Lacoste-Julien, Saga: A fast incremental gradient method with support for non-strongly convex composite objectives, in: Proceedings of Advances in Neural Information Processing Systems, Montréal, Canda, 2014, pp. 1646–1654.

[14] M. Schmidt, N. Le Roux, F. Bach, Minimizing finite sums with the stochastic average gradient, Mathematical Programming (2013) 1–30.

[15] J. Konecnỳ, P. Richtárik, Semi-stochastic gradient descent methods, arXiv preprint arXiv:1312.1666 2 (2.1) (2013) 3.

[16] L. Zhang, M. Mahdavi, R. Jin, Linear convergence with condition number independent access of full gradients, in: Proceedings of Advances in Neural Information Processing Systems, Lake Tahoe, USA, 2013, pp. 980–988.

[17] Z. Allen-Zhu, Y. Yuan, Improved svrg for non-strongly-convex or sum-of-non-convex objectives, in: Proceedings of International Conference on Machine Learning, New York, 2016.

[18] Z. Shen, H. Qian, T. Zhou, T. Mu, Adaptive Variance Reducing for Stochastic Gradient Descent., in: Proceedings of International Joint Conference on Artificial Intelligence, 2016.

[19] J. Konečný, J. Liu, P. Richtárik, M. Takáč, Mini-batch semi-stochastic gradient descent in the proximal setting, IEEE Journal of Selected Topics in Signal Processing 10 (2) (2014) 242–255.

[20] C. Tan, S. Ma, Y. H. Dai, Y. Qian, Barzilai-borwein step size for stochastic gradient descent, in: Proceedings of Advances in Neural Information Processing Systems, 2016.

[21] J. Philbin, O. Chum, M. Isard, J. Sivic, A. Zisserman, Object retrieval with large vocabularies and fast spatial matching, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2007.

[22] M. Perd, O. Chum, J. Matas, Efficient representation of local geometry for large scale object retrieval, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 9–16.

[23] A. Torii, J. Sivic, M. Okutomi, T. Pajdla, Visual place recognition with repetitive structures, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[24] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images.

[25] GriffinGS, HolubAD, PeronaP, Caltech-256 object category dataset, California Institute of Technology.

[26] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.