frontiers in Robotics and AI

ORIGINAL RESEARCH
published: 27 August 2019
doi: 10.3389/frobt.2019.00077

Incremental and Parallel Machine Learning Algorithms With Automated Learning Rate Adjustments

Kazuhiro Hishinuma[1]**** and Hideaki Iiduka***[2]

[1] Computer Science Program, Graduate School of Science and Technology, Meiji University, Kawasaki, Japan, [2] Department of Computer Science, Meiji University, Kawasaki, Japan

The existing machine learning algorithms for minimizing the convex function over a closed convex set suffer from slow convergence because their learning rates must be determined before running them. This paper proposes two machine learning algorithms incorporating the line search method, which automatically and algorithmically finds appropriate learning rates at run-time. One algorithm is based on the incremental subgradient algorithm, which sequentially and cyclically uses each of the parts of the objective function; the other is based on the parallel subgradient algorithm, which uses parts independently in parallel. These algorithms can be applied to constrained nonsmooth convex optimization problems appearing in tasks of learning support vector machines without adjusting the learning rates precisely. The proposed line search method can determine learning rates to satisfy weaker conditions than the ones used in the existing machine learning algorithms. This implies that the two algorithms are generalizations of the existing incremental and parallel subgradient algorithms for solving constrained nonsmooth convex optimization problems. We show that they generate sequences that converge to a solution of the constrained nonsmooth convex optimization problem under certain conditions. The main contribution of this paper is the provision of three kinds of experiment showing that the two algorithms can solve concrete experimental problems faster than the existing algorithms. First, we show that the proposed algorithms have performance advantages over the existing ones in solving a test problem. Second, we compare the proposed algorithms with a different algorithm Pegasos, which is designed to learn with a support vector machine efficiently, in terms of prediction accuracy, value of the objective function, and computational time. Finally, we use one of our algorithms to train a multilayer neural network and discuss its applicability to deep learning.

Keywords: support vector machines, neural networks, nonsmooth convex optimization, incremental subgradient algorithm, parallel subgradient algorithm, line search algorithm, parallel computing

OPEN ACCESS

Edited by:
Bologna Guido,
Université de Genève, Switzerland

Reviewed by:
Önder Tutsoy,
Adana Science and Technology University, Turkey
Narin Petrot,
Naresuan University, Thailand

****Correspondence:***
Kazuhiro Hishinuma
kaz@cs.meiji.ac.jp

Specialty section:
This article was submitted to Computational Intelligence, a section of the journal Frontiers in Robotics and AI

Received: *03 December 2018*
Accepted: *08 August 2019*
Published: *27 August 2019*

Citation:
Hishinuma K and Iiduka H (2019) Incremental and Parallel Machine Learning Algorithms With Automated Learning Rate Adjustments. Front. Robot. AI 6:77. doi: 10.3389/frobt.2019.00077

1. INTRODUCTION

In this paper, we consider a technique to adjust the learning rates that appear in subgradient algorithms for letting a generated sequence converge to an optimal solution. The subgradient algorithm (Bertsekas et al., 2003, section 8.2) and its variants (Nedić and Bertsekas, 2001; Shalev-Shwartz et al., 2011; Hishinuma and Iiduka, 2015) have been proposed as ways of solving the problem of minimizing a nonsmooth, convex function over a closed convex set by iterative processes like the steepest descent method for dealing with a smooth, convex function. These methods iterate the current approximate solution by shifting it along a descent direction at that point by a given degree called a *learning rate*. Although descent directions are decided on the basis of the subgradient at each point, the learning rates are generally decided for theoretical reasons for ensuring the convergence of the generated sequence. This implies that subgradient algorithms can be run more efficiently if we can choose more suitable learning rates concerning the objective function at each iteration. Therefore, we should consider how to choose better learning rates while at the same time maintaining the convergence properties.

We can reduce a lot of practical problems to ones solvable with subgradient algorithms, that is, problems of minimizing a nonsmooth, convex function over a closed convex set. One of the important applications is learning with a support vector machine. Support vector machines are effective and popular classification learning tools (Leopold and Kindermann, 2002; Lin et al., 2002; Pradhan et al., 2005; Shalev-Shwartz et al., 2011). The task of learning with a support vector machine is cast as an empirical loss minimization with a penalty term for the norm of the classifier that is being learned (Shalev-Shwartz et al., 2011, Problem 1). If this loss objective function is convex, we can handle this learning task by minimizing a nonsmooth, convex function over a closed convex set. There are practical optimization algorithms for solving this minimization problem, such as Pegasos (Shalev-Shwartz et al., 2011), the incremental subgradient algorithm (Nedić and Bertsekas, 2001), and the parallel subgradient algorithm (Hishinuma and Iiduka, 2015). These algorithms are variants of the subgradient algorithm. They iteratively choose training examples and improve their approximation by using a part of the objective function which corresponds to the chosen examples. Not limited to machine learning, there exist many applications of minimizing a nonsmooth, convex function over a closed convex set, such as signal recovery (Combettes, 2003), bandwidth allocation (Iiduka, 2013), and beamforming (Slavakis and Yamada, 2007). Hence, making the performance of these algorithms better would increase the efficiency of these applications. Here, we attempt to do so by modifying the selection of the learning rate.

Pegasos is a stochastic subgradient algorithm with a carefully chosen learning rate that is designed for efficiently learning with a support vector machine (Shalev-Shwartz et al., 2011). This learning rate is determined from the regularization constant of the penalty term. Hence, this algorithm can improve approximate solutions without having to adjust their learning rates for each individual learning task. However, it is specialized to learning with a support vector machine, and it cannot be applied to other applications such as deep learning.

The sequential minimal optimization (SMO) algorithm (Platt, 1998) is also used for learning with a support vector machine. This algorithm can be applied to a quadratic programming optimization problem appearing in learning with a dual form of a support vector machine and can solve it with a small amount of memory and quickly (Platt, 1998; Cristianini and Shawe-Taylor, 2000). However, this algorithm deals with the dual form of the optimization problem; as such, the number of objective variables is likely to be large when many instances are given to the learning task. Furthermore, the class of problem that this algorithm can deal with is limited to quadratic programming. This implies that it cannot be applied to general nonsmooth, convex programming.

In the field of mathematical optimization, the incremental and parallel subgradient algorithms (Nedić and Bertsekas, 2001; Hishinuma and Iiduka, 2015) are useful for solving problems involving the minimization of a nonsmooth, convex function over a closed convex set. The incremental subgradient algorithm (Nedić and Bertsekas, 2001) minimizes the objective function by using alternately one of the functions composing the summed objective, while the parallel subgradient algorithm (Hishinuma and Iiduka, 2015) minimizes it by using all of the composing functions independently. Since the parallel subgradient algorithm treats each of the composing functions independently, computations with respect to each function can be parallelized. It is expected that parallelization shortens the computational time of learning. This implies that the parallel subgradient algorithm can learn support vector machines for larger datasets and/or in a shorter time compared with other algorithms.

A weak point of the incremental and parallel subgradient algorithms (Nedić and Bertsekas, 2001; Hishinuma and Iiduka, 2015) is that they need to have suitably adjusted learning rates in order to run efficiently. However, the suitable learning rate depends on various factors, such as the number of the composing objective functions, number of dimensions, the shape of each objective function and constraint set, and the selection of subgradients. This implies that it is too difficult to choose a suitable learning rate before run-time. In contrast, Pegasos (Shalev-Shwartz et al., 2011) uses a concrete learning rate optimized for the task of learning with a support vector machine and does not require this learning rate to be adjusted. Therefore, it can be used more easily than the incremental and parallel subgradient algorithms (Nedić and Bertsekas, 2001; Hishinuma and Iiduka, 2015).

In unconstrained minimization algorithms, line searches are used to select a suitable learning rate (Hare and Lucet, 2014; Yuan et al., 2016). In particular, the *Wolfe conditions* (Wolfe, 1969) are learning rate criteria for the line search. The Wolfe conditions are such that the learning rate must satisfy a sufficient decrease condition and a curvature condition (Nocedal and Wright, 2006, Chapter 3). The sufficient decrease condition is that the learning rate is acceptable only if its function value is below a linear function with a negative slope. This condition ensures that the algorithms update an approximation to a better one. However, it is not enough to ensure that the algorithm makes reasonable progress because it will do so for all sufficiently small learning rates. Therefore, a curvature condition is invoked that generates a sequence further enough along the chosen direction.

Motivated by the idea of the line search, this paper proposes novel incremental and parallel subgradient algorithms that can run efficiently without precise learning rate adjustments. Cruz and Oliveira (2016) describes a gradient-projection algorithm with a line search that minimizes the objective function.

However, this algorithm assumes that the objective function is differentiable. In addition, it is designed for single-core computing; it is not useful in multi-core computing. Beltran and Heredia (2005) proposes the radar subgradient algorithm, which is a variant of the subgradient algorithm including a procedure for finding an effective learning rate by using a line search at each iteration. The line search method used in Beltran and Heredia (2005) is inspired by the cutting-plane method and works out a learning rate with the first-order information. However, this algorithm deals with the whole objective function and cannot use a part of the objective function at each iteration. This implies that it cannot be used in applications that give information to the algorithm through a data stream. In addition, the line search method used in Beltran and Heredia (2005) may fail and is distinct from the line search proposed in this paper. Hence, combining this line search method with the one we propose may have a complementary effect when the properties of the optimization problem are disadvantageous to one of the algorithms. Iiduka (2016b) gives an algorithm for solving fixed point problems, covering the constrained minimization problem discussed in this paper, with a line search. This algorithm has a fast convergence property, though it decides only the coefficient of the convex combination and is not designed for multi-core computing. The algorithm in Nedić and Bertsekas (2001), Hishinuma and Iiduka (2015), Iiduka (2015b, 2016a), and Hayashi and Iiduka (2018) requires a suitable learning rate in order to converge efficiently. However, as we mentioned before, the learning rate is very difficult to adjust.

In contrast to previous reports, this paper proposes incremental and parallel subgradient algorithms with a line search to find better learning rates than the ones used in the existing algorithms. To realize this proposal, we extend the concept of the *learning rate* to a *step-range*, which is a set of candidates for the learning rate. The line search procedure is given a step-range and chooses the most suitable learning rate among it at run-time. Using a line search with a step-range has three merits. First, the suitable learning rates chosen by the line search accelerate the algorithms and make their solutions better. Section 5 shows that the proposed algorithms gave better solutions than the one given by Pegasos (Shalev-Shwartz et al., 2011) when they all ran the same number of iterations. The second merit is that we do not need to adjust the learning rate precisely. The existing incremental and parallel subgradient algorithms (Nedić and Bertsekas, 2001; Hishinuma and Iiduka, 2015) cannot converge efficiently without appropriate adjustments to their learning rates. This is their weak point in comparison with Pegasos (Shalev-Shwartz et al., 2011). In contrast, the proposed algorithms only need step-ranges, i.e., rough candidates, to converge efficiently, because the line search automatically chooses the learning rates from among the step-range. Hence, they can be easily used to learn support vector machines. Finally, the proposed algorithms can be applied to difficult problems whose suitable learning rates cannot be chosen beforehand. Section 4 provides a condition on the step-range compositions to ensure they converge to an optimizer of the problem. Hence, even if a suitable learning rate cannot be specified beforehand, the line search can algorithmically find one at run-time and make the algorithms converge efficiently to an optimizer. We show that our algorithms converge to an optimizer to the problem when the step-range is diminishing. In addition, if the step-range is a singleton set, they coincide with the existing incremental and parallel subgradient algorithms (Nedić and Bertsekas, 2001; Hishinuma and Iiduka, 2015). Hence, the step-range is a generalization of the learning rates used in the existing algorithms.

We compared the proposed algorithms with Pegasos (Shalev-Shwartz et al., 2011) and the SMO algorithm on various datasets (LeCun et al., 1998b; Dheeru and Karra Taniskidou, 2017; Lin, 2017) for binary and multiclass classification. The results of the comparison demonstrated that the proposed algorithms perform better than the existing ones in terms of the value of the objective function for learning with a support vector machine and in terms of computational time. In particular, the parallel subgradient algorithm dramatically reduced the computational times of the learning tasks.

Stochastic subgradient algorithms are useful for learning with a multilayer neural network habitually (Bottou, 1991). The incremental subgradient algorithm is a specialization of the stochastic subgradient algorithm. Therefore, we can use one of our algorithms, a variant of the incremental subgradient algorithm, to train a multilayer neural network. We compared it with two other variants of the incremental subgradient algorithm. The results show that our algorithm can minimize the objective function of the trained neural network more than the others. This ability implies that it is also useful for training not only SVMs but also neural networks, including ones for deep learning.

This paper is organized as follows. Section 2 gives the mathematical preliminaries and mathematical formulation of the main problem. Section 3 presents our algorithms. We also show the fundamental properties of these algorithms that are used to prove the main theorems. Section 4 presents convergence analyses. Section 5 describes numerical comparisons of the proposed algorithms with the existing ones in Nedić and Bertsekas (2001), Shalev-Shwartz et al. (2011), and Hishinuma and Iiduka (2015) using concrete machine learning datasets (LeCun et al., 1998b; Dheeru and Karra Taniskidou, 2017; Lin, 2017). In this section, we also describe how to use one of the proposed algorithms to train a multilayer neural network for recognizing handwritten digits. Section 6 concludes this paper.

2. MATHEMATICAL PRELIMINARIES

Let $\mathbb{R}^N$ be an N-dimensional Euclidean space with the standard Euclidean inner product $\langle \cdot, \cdot \rangle : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ and its induced norm defined by $\|x\| := \langle x, x \rangle^{\frac{1}{2}}$. We define the notation $\mathbb{N} := \{1, 2, \ldots\}$ as the set of all natural numbers. Let $x_n \to x$ denote that the sequence $\{x_n\} \subset \mathbb{R}^N$ converges to a point $x \in \mathbb{R}^N$.

A *subgradient* g of a convex function $f : \mathbb{R}^N \to \mathbb{R}$ at a point $x \in \mathbb{R}^N$ is defined by $g \in \mathbb{R}^N$ such that $f(x) + \langle y - x, g \rangle \leq f(y)$ for all $y \in \mathbb{R}^N$. The set of all subgradients at a point $x \in \mathbb{R}^N$ is denoted as $\partial f(x)$ (Rockafellar, 1970, section 7.3, Takahashi, 2009).

The metric projection onto a nonempty, closed convex set $C \subset \mathbb{R}^N$ is denoted by $P_C : \mathbb{R}^N \to C$ and defined by

$\|x - P_C(x)\| = \inf_{y\in C}\|x - y\|$ (Bauschke and Combettes, 2011, section 4.2, Chapter 28). P_C satisfies the nonexpansivity condition (Takahashi, 2009, subchapter 5.2); i.e., $\|P_C(x) - P_C(y)\| \leq \|x - y\|$ for all $x, y \in \mathbb{R}^N$.

2.1. Main Problem

We will consider the following optimization problem (Nedić and Bertsekas, 2001; Hishinuma and Iiduka, 2015): let $f_i : \mathbb{R}^N \to [0,\infty) \quad (i = 1, 2, \dots, K)$ be convex, continuous functions and let C be a nonempty, closed convex subset of $\mathbb{R}^N$. Then,

$$\text{minimize } f(x) := \sum_{i=1}^{K} f_i(x) \quad \text{subject to } x \in C. \tag{1}$$

Let us discuss Problem (1) in the situation that a closed convex subset C of an N-dimensional Euclidean space $\mathbb{R}^N$ is simple in the sense that P_C can be computed within a finite number of arithmetic operations. Examples of a simple, closed convex set C are a closed ball, a half-space, and the intersection of two half-spaces (Bauschke and Combettes, 2011, Examples 3.16 and 3.21, and Proposition 28.19).

The task of learning with a support vector machine can be cast as Problem (1) (Shalev-Shwartz et al., 2011, Problem 1). Furthermore, there are a lot of applications not limited to learning with a support vector machine when f is nonsmooth but convex on $\mathbb{R}^N$ and when $C \subset \mathbb{R}^N$ is simple. For example, minimizing the total variation of a signal over a convex set and Tykhonov-like problems with L^1-norms (Combettes and Pesquet, 2007, I. Introduction) are able to be handled as Problem (1). Application of Problem (1) to learning with a support vector machine will be described in section 5.

The following assumptions are made throughout this paper.

Assumption 1 (Subgradient Boundedness; Nedić and Bertsekas, 2001, Assumption 2.1). *For all $i = 1, 2, \dots, K$, there exists $M_i \in (0,\infty)$ such that*

$$\|g\| \leq M_i \quad (x \in C; g \in \partial f_i(x)).$$

We define a constant $M := \sum_{i=1}^{K} M_i$.

Assumption 2 (Existence of Optimal Solution; Nedić and Bertsekas, 2001, Proposition 2.4). $\operatorname{argmin}_{x\in C} f(x) \neq \emptyset$.

3. PROPOSED ALGORITHMS AND THEIR FUNDAMENTAL PROPERTIES

3.1. Incremental Subgradient Algorithm

This subsection presents the incremental subgradient algorithm, Algorithm 1, for solving Problem (1).

Let us compare Algorithm 1 with the existing one (Nedić and Bertsekas, 2001). The difference is Step 6 of Algorithm 1. The learning rate λ_n of the existing algorithm must be decided before the algorithm runs. However, Algorithm 1 only needs the step-range $[\underline{\lambda}_n, \overline{\lambda}_n]$. A learning rate within the range used by Algorithm 1 can be automatically determined at run-time. Algorithm 1 coincides with the incremental subgradient algorithm when the given step-range $[\underline{\lambda}_n, \overline{\lambda}_n]$ is a singleton set, i.e., $\underline{\lambda}_n := \overline{\lambda}_n := \lambda_n$, which means that it is a generalization of the algorithm in Nedić and Bertsekas (2001). In this case, Algorithm 1 chooses only one learning rate λ_n from the singleton step-range $[\underline{\lambda}_n, \overline{\lambda}_n] = \{\lambda_n\}$.

Algorithm 1: Incremental subgradient algorithm

```
Require: $\forall n \in \mathbb{N}, [\underline{\lambda}_n, \overline{\lambda}_n] \subset (0,\infty)$.
1:  $n \leftarrow 1, x_1 \in C$.
2:  loop
3:      $y_{n,0} := x_n$.
4:      for $i = 1, 2, \dots, K$ do                    ▷ In sequence
5:          $g_{n,i} \in \partial f_i(y_{n,i-1})$.
6:          $\lambda_{n,i} \in [\underline{\lambda}_n, \overline{\lambda}_n]$.      ▷ By a line search algorithm
7:          $y_{n,i} := P_C(y_{n,i-1} - \lambda_{n,i} g_{n,i})$.
8:      end for
9:      $x_{n+1} := y_{n,K}$.
10:     $n \leftarrow n + 1$.
11: end loop
```

This difference has three merits. First, the suitably chosen learning rates in Step 6 accelerate convergence and make the solutions more accurate. Second, Algorithm 1 does not require the learning rate to be precisely adjusted in order for it to converge efficiently, unlike the existing incremental subgradient algorithm (Nedić and Bertsekas, 2001). Instead, Algorithm 1 only needs a rough step-range as the line search automatically chooses learning rates from among this range. Hence, it can easily be used to learn support vector machines. Finally, Algorithm 1 can be applied to problems in which a suitable learning rate cannot be chosen beforehand. Hence, even if the suitable learning rate cannot be specified, line search can algorithmically find this learning rate and make proposed algorithms converge efficiently to an optimizer.

Algorithm 1 has the following property, which is used for proving the main theorem in section 4. We omit the proof of this lemma here and refer the reader to (Hishinuma and Iiduka, 2016, Lemma 1).

Lemma 1 (Fundamental Properties of Algorithm 1; Hishinuma and Iiduka, 2016, Lemma 1). *Let $\{x_n\}$ be a sequence generated by Algorithm 1. Then, for all $y \in C$ and for all $n \in \mathbb{N}$, the following inequality holds:*

$$\|x_{n+1} - y\|^2 \leq \|x_n - y\|^2 - 2\sum_{i=1}^{K} \lambda_{n,i}(f_i(x_n) - f_i(y)) + \overline{\lambda}_n^2 M^2.$$

3.2. Parallel Subgradient Algorithm

Algorithm 2 below is an extension of the parallel subgradient algorithm Hishinuma and Iiduka (2015).

The difference between Algorithm 2 and the algorithm in Hishinuma and Iiduka (2015) is Step 5 of Algorithm 2. The existing algorithm uses a given learning rate λ_n, while

Algorithm 2: Parallel subgradient algorithm

```
Require: $\forall n \in \mathbb{N}, [\underline{\lambda}_n, \overline{\lambda}_n] \subset (0, \infty)$.
1: $n \leftarrow 1, x_1 \in C$.
2: loop
3:     for all $i \in \{1, 2, \dots, K\}$ do                ▷ Independently
4:         $g_{n,i} \in \partial f_i(x_n)$.
5:         $\lambda_{n,i} \in [\underline{\lambda}_n, \overline{\lambda}_n]$.        ▷ By a line search algorithm
6:         $y_{n,i} := P_C(x_n - \lambda_{n,i} g_{n,i})$.
7:     end for
8:     $x_{n+1} := \frac{1}{K} \sum_{i=1}^{K} y_{n,i}$.
9:     $n \leftarrow n + 1$.
10: end loop
```

Algorithm 2 chooses a learning rate λ_n from the step-range $[\underline{\lambda}_n, \overline{\lambda}_n]$ at run-time.

The common feature of Algorithm 2 and the parallel subgradient algorithm (Hishinuma and Iiduka, 2015) is loop independence (Step 3). This loop structure is not influenced by the computation order. Hence, each iteration of this loop can be computed in parallel. Therefore, parallelization using multi-core processing should be able to reduce the time needed for computing this loop procedure. Generally speaking, the main loop of Algorithm 2 is computationally heavier than the other subgradient algorithms including Pegasos, because it appends the learning rate selection (line search) procedure to the existing one. However, parallelization alleviates this effect of the line search procedure (This is shown in section 5).

Next, we have the following lemma.

Lemma 2 (Fundamental Properties of Algorithm 2; Hishinuma and Iiduka, 2016, Lemma 2). *Let $\{x_n\}$ be a sequence generated by Algorithm 2. Then, for all $y \in C$ and for all $n \in \mathbb{N}$, the following inequality holds:*

$$\|x_{n+1} - y\|^2 \leq \|x_n - y\|^2 - \frac{2}{K} \sum_{i=1}^{K} \lambda_{n,i}(f_i(x_n) - f_i(y)) + \overline{\lambda}_n^2 M^2.$$

3.3. Line Search Algorithms

Step 6 of Algorithm 1 and Step 5 of Algorithm 2 are implemented as line searches. The algorithms decide an efficient learning rate λ_n in $[\underline{\lambda}_n, \overline{\lambda}_n]$ by using $y_{n,i-1}$ in Algorithm 1 (or x_n in Algorithm 2), $g_{n,i}$, f_i, and other accessible information on i. This is the principal idea of this paper. We can use any algorithm that satisfies the above condition. The following are such examples.

The simplest line search is the *discrete argmin*, as shown in Algorithm 3.

First, we set the ratio candidates $\{L_1, L_2, \dots, L_k\} \subset [0, 1]$. In each iteration, we compute all of the candidate objectives for the learning rate $\lambda_{n,i} = L_t \overline{\lambda}_n + (1 - L_t)\underline{\lambda}_n \quad (t = 1, 2, \dots, k)$ and take the best one.

Algorithm 4 is a line search based on the Wolfe conditions.

It finds a learning rate that satisfies the sufficient decrease condition with logarithmic grids. Once this learning rate has been found, the algorithm stops and the learning rate it found is used in the caller algorithm. However, this algorithm may fail

Algorithm 3: Discrete argmin line search algorithm

```
1: $x_p := \begin{cases} y_{n,i-1} & \text{(Algorithm 1)}, \\ x_n & \text{(Algorithm 2)} \end{cases}$.
2: $\lambda_{n,i} \leftarrow L_1 \overline{\lambda}_n + (1 - L_1)\underline{\lambda}_n$.
3: for $L_t \in \{L_2, L_3, \dots, L_k\}$ do
4:     $t \leftarrow L_t \overline{\lambda}_n + (1 - L_t)\underline{\lambda}_n$.
5:     if $f_i(P_C(x_p - t g_{n,i})) < f_i(P_C(x_p - \lambda_{n,i} g_{n,i}))$ then
6:         $\lambda_{n,i} \leftarrow t$
7:     end if
8: end for
```

Algorithm 4: Logarithmic-interval armijo line search

```
1: $x_p := \begin{cases} y_{n,i-1} & \text{(Algorithm 1)}, \\ x_n & \text{(Algorithm 2)} \end{cases}$.
2: for $I_R = 1, 1/a, 1/a^2, \dots, 1/a^k$ do
3:     $\lambda_{n,i} \leftarrow I_R \overline{\lambda}_n + (1 - I_R)\underline{\lambda}_n$.
4:     if $f_i(P_C(x_p - \lambda_{n,i} g_{n,i})) \leq f_i(x_p) - c_1 \langle x_p - P_C(x_p - \lambda_{n,i} g_{n,i}), g_{n,i} \rangle$ then
5:         stop (success).
6:     end if
7: end for
8: stop (failed).
```

(Step 8). To avoid such a failure, we can make the caller algorithm use $\underline{\lambda}_n$. This is the largest learning rate of the candidates for making an effective update of the solution. The results of the experiments described in section 5 demonstrate effectiveness of this algorithm[1].

4. CONVERGENCE ANALYSIS

4.1. Sequence Convergence Theorem

Here, we first show that the limit inferiors of $\{f(x_n)\}$ generated by Algorithms 1 and 2 are equal to the optimal value of the objective function f. Next, we show that the generated sequence $\{x_n\}$ converges to a solution of Problem (1). The following assumption is used to show convergence of Algorithms 1 and 2.

Assumption 3 (Step-Range Compositions).

$$\sum_{n=1}^{\infty} \overline{\lambda}_n = \infty, \ \sum_{n=1}^{\infty} \overline{\lambda}_n^2 < \infty, \ \lim_{n \to \infty} \frac{\underline{\lambda}_n}{\overline{\lambda}_n} = 1, \ \sum_{n=1}^{\infty} (\overline{\lambda}_n - \underline{\lambda}_n) < \infty.$$

The following lemma states that some subsequence of the objective function value of the generated sequence converges to the optimal value. This lemma is used to prove the main theorem described next.

[1]In this case, i.e., when we use Algorithm 1 or Algorithm 2 with Algorithm 4, we have to give a step-range $[\underline{\lambda}_n, \overline{\lambda}_n]$, a constant c_1 appearing in the Armijo condition, a common ratio a and the number of trials k of Algorithm 4 as hyperparameters.

Lemma 3 (Evaluation of the Limit Inferior; Hishinuma and Iiduka, 2016, Lemma 3). *For a sequence $\{x_n\}$, if there exists $\alpha \in (0,\infty)$ such that, for all $y \in C$ and for all $n \in \mathbb{N}$,*

$$\|x_{n+1} - y\|^2 \leq \|x_n - y\|^2 - 2\alpha \sum_{i=1}^{K} \lambda_{n,i}(f_i(x_n) - f_i(y)) + \overline{\lambda}_n^2 M^2, \tag{2}$$

then,

$$\varliminf_{n\to\infty} f(x_n) = \min_{x\in C} f(x).$$

The following is the main theorem of this paper.

Theorem 1 (Main Theorem). *The sequence $\{x_n\}$ generated by Algorithm 1 or 2 converges to an optimal solution to the main problem (1).*

Proof: Let $\hat{y} \in \operatorname{argmin}_{x\in C} f(x)$ and fix $n \in \mathbb{N}$. From Lemmas 1 and 2, there exists $\alpha \in (0,\infty)$ such that

$$\|x_{n+1} - \hat{y}\|^2 \leq \|x_n - \hat{y}\|^2 - 2\alpha \sum_{i=1}^{K} \lambda_{n,i}(f_i(x_n) - f_i(\hat{y})) + \overline{\lambda}_n^2 M^2.$$

From $0 \leq f_i(\hat{y}), f_i(x_n) \quad (i = 1, 2, \ldots, K)$, we have

$$\begin{aligned}
\|x_{n+1} - \hat{y}\|^2 &\leq \|x_n - \hat{y}\|^2 - 2\alpha \underline{\lambda}_n \sum_{i=1}^{K} f_i(x_n) \\
&\quad + 2\alpha \overline{\lambda}_n \sum_{i=1}^{K} f_i(\hat{y}) + \overline{\lambda}_n^2 M^2 \\
&= \|x_n - \hat{y}\|^2 - 2\alpha \underline{\lambda}_n \sum_{i=1}^{K} (f_i(x_n) - f_i(\hat{y})) \\
&\quad + 2\alpha(\overline{\lambda}_n - \underline{\lambda}_n) \sum_{i=1}^{K} f_i(\hat{y}) + \overline{\lambda}_n^2 M^2 \\
&\leq \|x_n - \hat{y}\|^2 + 2\alpha f(\hat{y})(\overline{\lambda}_n - \underline{\lambda}_n) + \overline{\lambda}_n^2 M^2 \qquad (3) \\
&\leq \|x_1 - \hat{y}\|^2 + 2\alpha f(\hat{y}) \sum_{i=1}^{n} (\overline{\lambda}_i - \underline{\lambda}_i) + M^2 \sum_{i=1}^{n} \overline{\lambda}_i^2.
\end{aligned}$$

From Assumption 3, the left side of the above inequality is bounded. Hence, $\{x_n\}$ is bounded. From Lemma 3, a subsequence $\{x_{n_i}\} \subset \{x_n\}$ and $u \in \operatorname{argmin}_{x\in C} f(x)$ exist such that $x_{n_i} \to u$. Using [Berinde, 2007, Lemma 1.7. (ii)] with inequality (3), this implies $x_n \to u$. This completes the proof. □

4.2. Convergence Rates

To show the convergence rates of Algorithms 1 and 2, we assume $\underline{\lambda}_n := \overline{\lambda}_n := 1/n$ for all $n \in \mathbb{N}$. We also assume the existence of $\mu \in (0,\infty)$ such that

$$f(x) - f(\hat{y}) \geq \mu \|x - \hat{y}\|^2 \quad \left(x \in C, \hat{y} \in \operatorname*{argmin}_{u\in C} f(u)\right). \tag{4}$$

The strong convexity of f implies Condition (4) [Nedić and Bertsekas, 2001, Inequality (16)]. First, We give the following lemma, which is required to show the convergence rates of Algorithms 1 and 2.

Lemma 4. (Nedić and Bertsekas, 2001, Lemma 2.1; Polyak, 1987, Lemma 4) *Let $\{u_n\} \subset [0,\infty)$ be such that*

$$u_{n+1} \leq \left(1 - \frac{p}{n}\right) u_n + \frac{d}{n^2} \quad (n \in \mathbb{N})$$

for some $p, d \in (0,\infty)$. Then

$$\begin{cases} u_n = O(\frac{1}{n^p}) & (p < 1), \\ u_n = O(\frac{\log n}{n}) & (p = 1), \\ u_n \leq \frac{d}{n(p-1)} + o(\frac{1}{n}) & (p > 1). \end{cases}$$

Next, we prove two propositions that show the convergence rates of Algorithms 1 and 2.

Proposition 1 (Convergence Rate of Algorithm 1). *Let $\{x_n\}$ be a sequence generated by Algorithm 1 and $\hat{y} \in \operatorname{argmin}_{y\in C} f(y)$. Then, the following hold:*

$$\begin{cases} \|x_{n+1} - \hat{y}\| = O(\frac{1}{n^{2\mu}}) & (2\mu < 1), \\ \|x_{n+1} - \hat{y}\| = O(\frac{\log n}{n}) & (2\mu = 1), \\ \|x_{n+1} - \hat{y}\| \leq \frac{M^2}{n(2\mu-1)} + o(\frac{1}{n}) & (2\mu > 1). \end{cases}$$

Proof: From Lemma 1 and inequality (4), we have

$$\|x_{n+1} - \hat{y}\|^2 \leq \left(1 - \frac{2\mu}{n}\right) \|x_n - \hat{y}\|^2 + \frac{M^2}{n^2}.$$

for all $n \in \mathbb{N}$. Lemma 4 with $p := 2\mu, d := M^2$ completes the proof. □

This result implies that Algorithm 1 is in the same class of convergence efficiency as the incremental subgradient algorithm [Nedić and Bertsekas, 2001, Proposition 2.8].

Proposition 2 (Convergence Rate of Algorithm 2). *Let $\{x_n\}$ be a sequence generated by Algorithm 2 and $\hat{y} \in \operatorname{argmin}_{y\in C} f(y)$. Then, the following hold:*

$$\begin{cases} \|x_{n+1} - \hat{y}\| = O(\frac{1}{n^{2\mu/K}}) & \left(\mu < \frac{K}{2}\right), \\ \|x_{n+1} - \hat{y}\| = O(\frac{\log n}{n}) & \left(\mu = \frac{K}{2}\right), \\ \|x_{n+1} - \hat{y}\| \leq \frac{M^2}{n(2\mu/K-1)} + o(\frac{1}{n}) & \left(\mu > \frac{K}{2}\right). \end{cases}$$

Proof: From Lemma 2 and inequality (4), we have

$$\|x_{n+1} - y\|^2 \leq \left(1 - \frac{2\mu}{nK}\right) \|x_n - \hat{y}\|^2 + \frac{M^2}{n^2}.$$

for all $n \in \mathbb{N}$. Lemma 4 with $p := 2\mu/K, d := M^2$ completes the proof. □

To above analyses assumed $\underline{\lambda}_n = \overline{\lambda}_n$. However, Algorithms 1 and 2 can use different values of $\underline{\lambda}_n$ and $\overline{\lambda}_n$. This implies that Algorithms 1 and 2 may converge faster than theoretical rates given here.

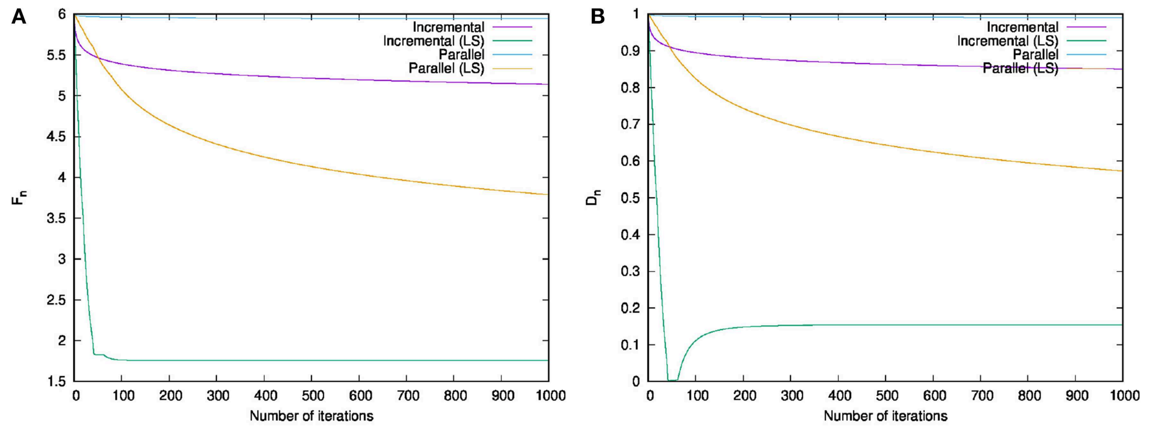

FIGURE 1 | Numerical comparison of running the existing and proposed algorithms on the test problem 1. **(A)** Behavior of F_n in each iteration n. **(B)** Behavior of D_n in each iteration n.

5. EXPERIMENTS

In this section, we present the results of experiments evaluating our algorithms and comparing them with the existing algorithms. For our experiments, we used a MacPro (Late 2013) computer with a 3 GHz 8-Core Intel Xeon E5 CPU, 32 GB, 1,866 MHz DDR3 memory, and 500 GB flash storage. The operating system was MacOS Sierra (version 10.12.6). The experimental codes were written in Python 3.6 and ran on the CPython implementation.

5.1. Validation of Convergence Properties With a Simple Problem

A concrete test problem with a closed-form solution is a good way to evaluate the performance of algorithms in detail (Tutsoy and Brown, 2016a,b). Here, we used the existing and proposed algorithms to solve a simple problem. The goals were to compare their performances under equal conditions, to use the best parameters for each algorithm calculated from the theoretical analyses, and to evaluate these algorithms with the detailed indicators such as the distance between an acquired solution and the actual solution of the test problem. The test problem is as follows.

Problem 1 (Test Problem). *Let* $f_i(x) := (i+1)x_i^2$ $(x \in \mathbb{R}^N; i = 1, 2, \dots, n)$, $c \in \mathbb{R}^N$, *and* $r \in \mathbb{R}$. *Then, we would like to*

$$\text{minimize}\, f(x) := \sum_{i=1}^{K} f_i(x)$$
$$\text{subject to } \|x - c\| \le r \text{ and } x_i = 0\ (i = 3, 4, \dots, N).$$

This problem is obviously an instance of Problem 1. Of course, the continuity of the objective function and the boundedness of the constraint ensure the above problem satisfies Assumption 1. We set $c := (2, 1, 0, \dots, 0)^\top$ and $r := 1$. The optimal solution is accordingly $x^\star = ((2 - /\sqrt{2})/2, (2 - \sqrt{2})/2, 0, \dots, 0)$.

We set the number of dimensions to $N := 16$, i.e., equal to the number of logical cores of the experimental computer. We gave $x_1 := (2, 1, 0, \dots, 0)^\top$, the center of the feasible set, as an initial point. We selected the incremental and parallel subgradient algorithms for comparison. These algorithms use a priori given learning rates; that is, they coincide with Algorithms 1, 2 with the settings $\underline{\lambda}_n = \overline{\lambda}_n = \lambda_n \in (0, \infty)$ $(n \in \mathbb{N})$. In this comparison, we gave learning rates of $\lambda_n := 1/(nN^2)$, which are appropriately chosen based on the following proposition related to the existing algorithms.

Proposition 3 (Iiduka, 2015a, Lemma 2.1; Yamada and Ogura, 2005, Lemma 3.1). *Suppose that* $f : \mathbb{R}^N \to \mathbb{R}$ *is c-strongly convex and differentiable,* $\nabla f : \mathbb{R}^N \to \mathbb{R}^N$ *is L-Lipschitz continuous, and* $\mu \in (0, 2c/L^2)$. *Define* $T : \mathbb{R}^N \to \mathbb{R}^N$ *by* $T(x) := x - \mu \nabla f(x)$ $(x \in \mathbb{R}^N)$. *Then, T is a contractive mapping.*

We set $\underline{\lambda}_n := 100/((n + 10,000)N^2), \overline{\lambda}_n := 100/(nN^2)$ for each $n \in \mathbb{N}$ as the parameters of the proposed algorithms. This step-range contains the learning rates of the existing algorithms. We used Algorithm 4 with the parameters $c_1 := 0.99$, $a := 0.5$, and $k := 7$. We limited the iterations to 1,000 and evaluated the following indicators:

- F_n: value of the objective function, i.e., $F_n := \sum_{i=1}^{n} f_i(x_n)$,
- D_n: distance to the optimal solution, i.e., $D_n := \|x^\star - x_n\|$,
- T_n: running time of the algorithm.

The behaviors of $\{F_n\}$ and $\{D_n\}$ in each iteration n are shown in **Figure 1**. The result of Algorithm 1 dropped to the optimal dramatically in terms of both $\{F_n\}$ and $\{D_n\}$ within the first fifty iterations. The graphs of the other algorithms decreased similarly, but those of the algorithms with the line search decreased faster. **Table 1** lists the running times of the existing and proposed algorithms for 1,000 iterations. Compared with the existing algorithms, the proposed algorithms needed a bit more time for running. However, they dramatically reduced the value of the objective function. Therefore, they converged faster that the existing algorithms.

TABLE 1 | Running time of each algorithm in solving the test problem 1.

Algorithm	Running time ($T_{1,000}$ [s])
Incremental subgradient algorithm	0.34341614
Algorithm 1	0.97763861
Parallel subgradient algorithm	0.11232432
Algorithm 2	0.24512658

5.2. Comparison of the Existing and Proposed Algorithms in the Task of Learning With Support Vector Machines

This subsection compares Algorithms 1 and 2 with Pegasos (Shalev-Shwartz et al., 2011). To evaluate their performance, we applied them to the following learning task.

Problem 2 (The task of learning with a support vector machine Shalev-Shwartz et al. (2011)). *Let C be a positive real number. Given a training set $\{(x_i, y_i)\}$, where $x_i \in \mathbb{R}^N \quad (i = 1, 2, \dots, K)$ and $y_i \in \{1, -1\} \quad (i = 1, 2, \dots, K)$, we would like to*

$$\text{minimize } f(w) := \frac{1}{C}\|w\|^2 + \frac{1}{K}\sum_{i=1}^{K}\max\{0, 1 - y_i\langle w, x_i\rangle\}$$
$$\text{subject to } w \in X := \{w : \|w\| \leq \sqrt{C}\}.$$

This optimization problem is introduced in Shalev-Shwartz et al. (2011) for learning with a support vector machine. The first term of the objective function is a penalty term that depends on the constraint set, and the second term is a loss function. The loss function returns higher values if the learner w can not classify an instance (x_i, y_i) correctly. The norm value of the learner w does not affect the classification results due to the immutability of the signs of the decision function $\langle w, x_i\rangle$. Therefore, we can limit this value to a constant C. Now, let $f_i(w) := ((1/C)\|w\|^2 + \max\{0, 1 - y_i\langle w, x_i\rangle\})/K$. Then, $f = \sum_{i=1}^{K} f_i$ holds and Problem 2 can be handled as an instance of Problem (1).

We used the machine learning datasets shown in **Table 2**. The "australian" data set is from LIBSVM Data (Lin, 2017). The "MNIST" data set contains handwritten "0" and "1" digits and is provided by LeCun et al. (1998b). The "RANDOM1" and "RANDOM2" datasets were generated using the `sklearn.datasets.make_classification` function with a fixed `random_state`. The others are from the UCI Machine Learning Repository (Dheeru and Karra Taniskidou, 2017). The number of classes of "iris (multiclass)" is three, and the others are binary classification datasets.

Missing values were complemented by using the `sklearn.impute.SimpleImputer` class. Categorical attributes were binarized using the `sklearn.preprocessing.OneHotEncoder` class. Each data set was scaled using the `sklearn.preprocessing.StandardScaler` class. These preprocessing methods and classes are from the scikit-learn (Pedregosa et al., 2011) package for Python3.

TABLE 2 | Datasets used in our experiments.

Name	#Instances	#Attributes	Missing values	Attribute characteristics
Iris (binary class)	100	4	No	Real
Iris (multiclass)	150	4	No	Real
Australian	590	14	No	Real
Horse-colic	368	27	Yes	Categorical, Integer, Real
Breast-cancer-wisconsin	699	10	Yes	Integer
Census-income	48,842	14	Yes	Categorical, Integer
Internet-advertisements	3,279	1,558	Yes	Categorical, Integer, Real
MNIST	14,780	784	No	Integer
RANDOM1	20	100	No	Real
RANDOM2	200	1,000	No	Real

The Pegasos algorithm used for this comparison is listed as Algorithm 5.

Algorithm 5: Pegasos (Shalev-Shwartz et al., 2011, **Figure 1**)

```
1: $n \leftarrow 1, w_1 \in X$.
2: loop
3:     $i_n \in \{1, 2, \dots, K\}$.        ▷ Chosen uniformly at random
4:     $g_n \in \partial f_{i_n}(w_n)$.
5:     $\lambda_n := C/n$.
6:     $w_{n+1} \leftarrow P_X(w_n - \lambda_n g_n)$.
7:     $n \leftarrow n + 1$.
8: end loop
```

We set $C := 10^{-1}$ and gave $\lambda_n := 10^{-1}/(n + 10^8)$ to Algorithms 1 and 2. We used Algorithm 4 with $c_1 := 0.99$ for the line search step in Algorithms 1 and 2. The main loops in Algorithms 1 and 5 were iterated $100K$ times, while the main loop in Algorithm 2 was iterated 100 times. This setting means that the algorithms could refer to each of the functions f_i $(i = 1, 2, \dots, K)$ 1,000 times.

We added scores of the SMO algorithm, one of the major algorithms for learning with a support vector machine, to the experimental results for each dataset. We used the implementation of the SMO algorithm in Python[2] for calculating these scores.

First, let us look at the results for the iris (binary class) data set. **Table 3** lists the computational times for learning, the classification scores on the training and test sets, and the values of the objective function. We used the `sklearn.model_selection.train_test_split` method provided by the scikit-learn package (Pedregosa et al., 2011) to split the dataset into training and test sets. The number of instances in the training set was 30 and the number of instances in the test set was 70. The results indicate that

[2]https://github.com/LasseRegin/SVM-w-SMO

TABLE 3 | Iris (binary class).

Algorithm	Time [sec]	Score (Training)	Score (Test)	Objective
Pegasos	0.12741225	1.00000000	1.00000000	0.95967555
Algorithm 1	0.28701049	1.00000000	1.00000000	0.94237229
Algorithm 2	0.03734168	1.00000000	1.00000000	0.91133305
SMO Algorithm	0.00515115	1.00000000	1.00000000	–

TABLE 4 | Iris (multiclass; Algorithms 1, 2, and 5 are used as solvers for the subproblem appearing in this multiclass classification experiment).

Algorithm	Time [sec]	Score (Training)	Score (Test)	Avg. Objective
Pegasos	0.59731907	0.77777778	0.79047619	0.98524879
Algorithm 1	1.30901892	0.77777778	0.80952381	0.97505393
Algorithm 2	0.08996129	0.80000000	0.81904762	0.95314453
SMO Algorithm	0.05340354	0.80000000	0.82857143	–

Algorithm 2 performed better than Pegasos and Algorithm 1 in terms of computational time and value of the objective function. In addition, Algorithm 1 worked out a better approximation than Pegasos did in terms of the objective function. Hence, Algorithms 1 and 2 ran more efficiently than the existing algorithm. However, the SMO algorithm ran more quickly than the other algorithms, while keeping the highest score.

Next, let us look at the results of the multiclass classification using the iris (multiclass) dataset. **Table 4** lists the computational times for learning and the classification scores on the training and test sets. We used the `sklearn.model_selection.train_test_split` method provided by the scikit-learn package (Pedregosa et al., 2011) to split the dataset into training and test sets. To construct multiclass classifiers from Algorithm 1, 2, and 5, we used `sklearn.multiclass.OneVsRestClassifier` class which provides a construction of one-versus-the-rest (OvR) multiclass classifiers. In this experiment, the number of instances in the training set was 45 and the number of instances in the test set was 105. The results show that Algorithms 1 and 2 performed better than Pegasos with respect to their scores for the training and test sets. In addition, the computational time of Algorithm 2 was shorter than those of Pegasos and Algorithm 1. In this case, Algorithm 2 learned a classifier whose classification score is similar to the one of the SMO algorithm in almost same running time.

To compare the algorithms in detail, we conducted experiments on other datasets: australian, horse-colic, breast-cancer-wisconsin, census-income, internet-advertisements, MNIST, RANDOM1, and RANDOM2. We performed a stratified five-fold cross-validation with the `sklearn.model_selection.StratifiedKFold` class. **Table 5** shows the averages of the computational times for learning, the classification scores on the test sets, and the values of the objective function for each dataset. TLE (time limit exceeded) in the table means that the experiment was compulsorily terminated because the running time of the SMO algorithm excessively exceeded those of the other algorithms. The classification scores are calculated using the following formula implemented as the `sklearn.base.ClassifierMixin.score` method,

$$(\text{Score}) := \frac{(\#\text{Accurate Instances})}{(\#\text{Instances})}.$$

This value is an increasing evaluation of goodness of fit (Pedregosa et al., 2011, section 4).

Let us evaluate the computational times for learning, the classification scores on the test sets, and the values of the objective function in order. For a detailed, fair, statistical comparison, we used an analysis of variance (ANOVA) test and Tukey–Kramer's honestly significant difference (HSD) test. We used the `scipy.stats.f_oneway` method in the SciPy library as the implementation of the ANOVA tests and the `statsmodels.stats.multicomp.pairwise_tukeyhsd` method in the StatsModels package as the implementation of Tukey–Kramer's HSD test. The ANOVA test examines whether the hypothesis that the given groups have the same population mean is rejected or not. Therefore, we can use it for finding an experimental result that has a significant difference. Tukey–Kramer's HSD test can be used to find specifically which pair has a significant difference in groups. We set 0.05 (5%) as the significance level for the ANOVA and Tukey–Kramer's HSD tests and used the results of each fold of the cross-validation for the statistical evaluations described below.

First, we consider the computation times for learning. All p-values computed by the ANOVA tests were much <0.05; this range was from 10^{-26} to 10^{-8}. This implies that a significant difference exists in terms of the computation time between the algorithms for every dataset. The results of the Tukey–Kramer's HSD tests showed that the computation times of Algorithm 2 for the australian, horse-colic, breast-cancer-wisconsin, census-income, internet-advertisements, and MNIST datasets were significantly shorter than those of Pegasos, Algorithm 1 and the SMO algorithm. However, the null hypotheses about Algorithm 2 and the SMO algorithm for the RANDOM1 dataset, and Algorithm 2 and Pegasos for the RANDOM2 dataset were not rejected. Therefore, for most of the practical datasets, Algorithm 2 runs significantly faster than the existing algorithms. However, it seems that there are a few cases where the computation time of the Algorithm 2 roughly equals those of the existing algorithms.

Next, we consider the classification scores on the test sets. The ANOVA tests indicate that significant differences may exist in the census-income, internet-advertisements, and RANDOM2 datasets. However, Tukey–Kramer's HSD test could not reject the null hypotheses between any two algorithms for the census-income dataset. The results of the Tukey–Kramer's HSD tests showed that the scores of Algorithm 1 were significantly worse than those of the other algorithms for the internet-advertisements and RANDOM2 datasets. Moreover, they showed that the scores of Algorithm 2 were significantly better than

TABLE 5 | Averages of computational times for learning, classification scores on the test sets, and values of the objective function for each dataset.

	Australian			Horse-colic		
Algorithm	**Time [sec]**	**Score**	**Objective**	**Time [sec]**	**Score**	**Objective**
Pegasos	2.42488674	0.82920957	0.99754775	1.54011672	0.71261261	0.99908892
Algorithm 1	6.04898934	0.84939531	0.99070582	3.97349780	0.70713213	0.99907467
Algorithm 2	0.06925168	0.85227300	0.95032527	0.05841917	0.72620120	0.96485216
SMO Algorithm	1.63863547	0.86238696	–	4.96065068	0.71193694	–

	Breast-cancer-wisconsin			Census-income		
Algorithm	**Time [sec]**	**Score**	**Objective**	**Time [sec]**	**Score**	**Objective**
Pegasos	2.49116932	0.96843767	0.99228738	180.51582360	0.70191638	0.99995947
Algorithm 1	6.12068390	0.96558053	0.95931909	456.40569000	0.71029029	0.99909114
Algorithm 2	0.06906789	0.96558053	0.82970374	0.52751211	0.71031078	0.96254982
SMO Algorithm	0.98688517	0.96989708	–	TLE	–	–

	Internet-advertisements			MNIST		
Algorithm	**Time [sec]**	**Score**	**Objective**	**Time [sec]**	**Score**	**Objective**
Pegasos	17.90985671	0.95730497	0.99954933	69.15901990	0.98687356	0.99894996
Algorithm 1	44.18707687	0.59436744	0.99972300	153.40252700	0.99032472	0.99827819
Algorithm 2	0.30783534	0.95517036	0.87879677	0.87765352	0.99269253	0.60622818
SMO Algorithm	TLE	–	–	TLE	–	–

	RANDOM1			RANDOM2		
Algorithm	**Time [sec]**	**Score**	**Objective**	**Time [sec]**	**Score**	**Objective**
Pegasos	0.34232559	0.88000000	0.99113491	3.99642010	0.84699855	0.99941910
Algorithm 1	1.38527165	0.86000000	0.99789224	17.21619467	0.68499385	0.99982958
Algorithm 2	0.03952229	0.90000000	0.93026020	0.04702928	0.87099875	0.95740116
SMO Algorithm	0.09197520	0.84000000	–	16.82656507	0.79700312	–

those of Algorithm 1 and the SMO algorithm for the RANDOM2 dataset. Although each algorithm may have advantages or disadvantages compared with the others on certain datasets, the classification scores of the four algorithms were roughly similar as a whole.

Next, we consider the values of the objective function. All p-values computed by the ANOVA tests were much <0.05; this range was from 10^{-32} to 10^{-12}. This implies that a significant difference exists in terms of the values of the objective function between the algorithms for every dataset. The results of the Tukey–Kramer's HSD tests showed that the values of the objective function of Algorithm 2 were significantly lower than those of Pegasos and Algorithm 1 for all datasets. Therefore, Algorithm 1 reduced the value of objective function more than the other algorithms.

Figure 2 illustrates a box-plot comparison of Pegasos, Algorithm 1, Algorithm 2, and the SMO Algorithm in terms of classification scores on the test sets. We used the results of all folds of the cross-validations and all datasets shown in **Table 5** for making this box-plot comparison. The horizontal lines in the boxes represent the median scores, and the boxes represent the upper and lower quartiles of the resulting scores. Similar to the above discussion of the average, we find that Algorithm 2 has the best median of the classification scores among the four algorithms. The results of Pegasos were similar to those of Algorithm 2; however, the computation time of Algorithm 2 was dramatically shorter than that of Pegasos. Therefore, box-plot comparison also shows that Algorithm 2 is the most useful method for learning with a support vector machine.

In conclusion, the above comparison indicates that, whichever algorithm we use, we can obtain classifiers whose classification abilities are similar. However, Algorithm 2 runs faster than the other algorithms, and it reduces the value of the objective function more. Therefore, the series of experiments and considerations lead us to conclude that Algorithm 2 is useful for learning with a support vector machine.

5.3. Application to Learning Multilayer Neural Networks

Let us consider using the proposed algorithms to learn a multilayer neural network with. Our algorithms are not limited to being used for learning support vector machines; they can also be used for optimizing general functions. Therefore, we can also use them for learning a multilayer neural network. Here, we should note that the incremental subgradient algorithm is a specialization of the stochastic subgradient algorithm, which is a useful algorithm for learning a neural network. Hence, we decided to apply it to a concrete task for learning a multilayer neural network and evaluate its applicability to learning deep neural networks.

We used the MNIST database (LeCun et al., 1998b) of handwritten digits for this experiment. The goal is recognizing what Arabic numerals are written on the given images. To achieve this goal, we can use 60,000 examples contained in the training set. Each example is composed of a 28×28 image that expresses a handwritten digit and its corresponding label that is an integer number from zero to nine. For the evaluation and comparison of the learning results, we used a test set containing 10,000 examples formatted in the same way.

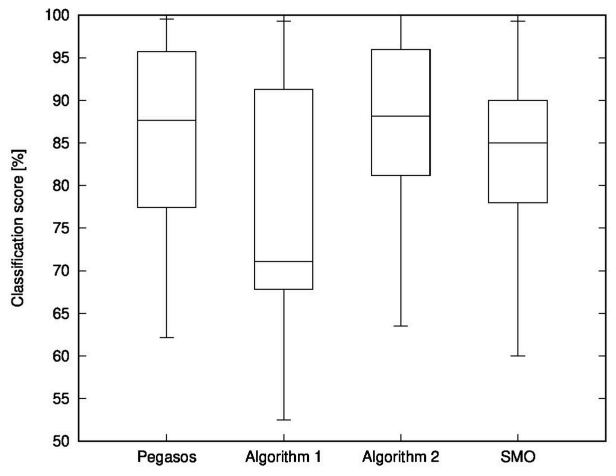

FIGURE 2 | Box-plot comparison of Pegasos, Algorithm 1, Algorithm 2, and SMO Algorithm in terms of classification scores on the test sets.

We constructed and trained a multilayer neural network shown in **Figure 3** for learning the MNIST database. We used three Affine layers with two ReLU (Rectified Linear Unit) activation functions and, for the output, a Softmax activation function. We used the cross-entropy error function as the objective function for training the neural networks.

An Affine layer $A_{W,b}$ transforms a given vector $x \in \mathbb{R}^n$ into

$$A_{W,b}(x) := Wx + b$$

with the parameter $W \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, where n is the number of dimensions of the input vector and m is the number of dimensions of the output vector. The first Affine layer transforms a $784 (= 28 \times 28)$-dimensional vector, which expresses a given image, into a 300-dimensional vector. The second Affine layer transforms a 300-dimensional vector into a 100-dimensional vector. The third Affine layer transforms a 100-dimensional vectors into a 10-dimensional vector, which expresses each probability that the given image is the corresponding number. We used the number of dimensions described in LeCun et al. (1998a) for each Affine layer.

The ReLU function transforms each element x_k ($k = 1, 2, \ldots, n$) of a given vector $x \in \mathbb{R}^n$ into $\max\{x_k, 0\}$. The Softmax function transforms a given vector $x := (x_k)_{k=1}^n \in \mathbb{R}^n$ as follows:

$$\mathrm{Softmax}(x) := \frac{1}{\sum_{k=1}^n e^{x_k}} (e^{x_1}, e^{x_2}, \ldots, e^{x_n})^\top.$$

We define the cross-entropy error $E : \mathbb{R}^n \to \mathbb{R}$, which is used as the objective function for training neural networks, as follows:

$$E(x) := -\sum_{k=0}^{9} y_k \log(x_k),$$

where the vector $x := (x_k)_{k=0}^9 \in \mathbb{R}^{10}$ is the output of the current neural network and y_k ($k = 0, 1, 2, \ldots, 9$) is one if the label is k and zero otherwise.

In this experiment, we wanted to minimize the cross-entropy error of the training dataset concerning the parameters W_k, b_k for each Affine layer $A_k (k = 1, 2, 3)$. The number of dimensions of the parameters is $784 \times 300 + 300 = 235,500$ for the first Affine layer, $300 \times 100 + 100 = 30,100$ for the second Affine layer, and $100 \times 10 + 10 = 1010$ for the third Affine layer. Hence, the

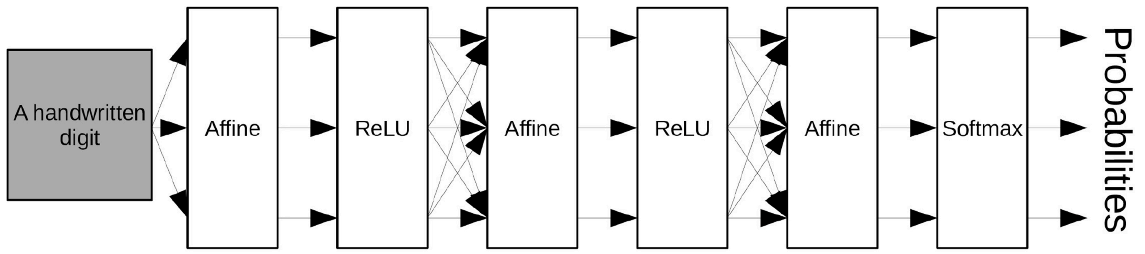

FIGURE 3 | Neural network diagram used to recognizing the MNIST handwritten digits.

total number of dimensions of the variables for this minimization problem is $235,500 + 30,100 + 1,010 = 266,610$.

We ran Algorithm 1 with the Discrete Argmin Line Search described in Algorithm 3 and compared its behaviors when we used a constant learning rate $\lambda_{n,i} := 0.1$, diminishing learning rate $\lambda_{n,i} := (0.1 \times 20)/n$, and learning rates found by the line search in the step-range $[(0.1 \times 20)/(n + 100), (0.1 \times 20)/n]$ for the number of iterations ($n = 1, 2, \ldots$). We set the coefficients for each step-size such that these upper bounds would be equal to each other when the algorithm exits. To use the proposed algorithm, we have to compute the subgradients of the objective function. Here, we used approximations of them worked out by the backpropagation algorithm.

We used the computer described in section 2.1 for these experiments. We wrote the experimental codes in Python 3.6.6 with the NumPy 1.15.4 library. We divided the datasets into 600 mini-batches, each of which contained 100 examples; in other words, we solved the problem to minimize the sum of 600 objective functions. We converted and flattened the handwritten digit images into vectors and divided their elements by 255 for regularization. The parameters for each Affine layer were initialized using a Gaussian distribution of mean zero and variance 0.01.

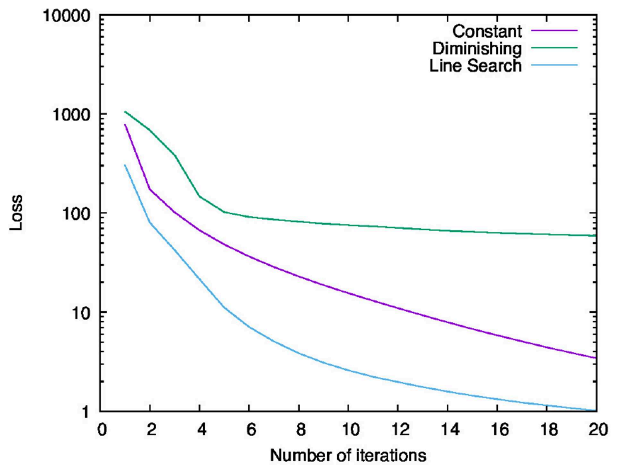

FIGURE 4 | Behavior of the values of the objective function for each iteration.

Figure 4 shows the behavior of the values of the objective function for each iteration. The violet line labeled "Constant" shows the result of using the constant learning rate, while the green line labeled "Diminishing" shows the result of using the diminishing learning rate, and the cyan line labeled "Line Search" shows that of using the learning rate computed with the line search. Overall, we can see that all the results decrease monotonously. This implies that Algorithm 1 can minimize the objective function with any of the above learning rate settings. The range of reduction of the result by using the diminishing learning rate is less than others. One possible reason is that learning rate becomes too small to minimize the objective function sufficiently. Indeed, from the second to fourth iteration, the result for the diminishing learning rate fell steeply, but this variation became smaller and smaller after the sixth iteration. In contrast to this result, the results for the constant learning rate and the learning rate computed with the line search minimized the objective function continuously and dramatically. In particular, we can see that the line search found the most efficient learning rates of these experiments.

Next, let us examine the classification accuracies. **Figure 5** shows the behavior of the classification accuracies for the training and test data. The left-hand graph (**Figure 5A**) shows the classification accuracies for the training data and the right-hand graph (**Figure 5B**) shows those for the test data. The legends of these graphs are the same as in **Figure 4**. We can see that all the results increased, heading for 100%. For both data, the score of "Line Search" is higher than others and the score of "Diminishing" is the lowest. This order is the same as what we saw in **Figure 2**. Therefore, using the learning rates computed by the line search makes us able to minimize the objective function most and to achieve

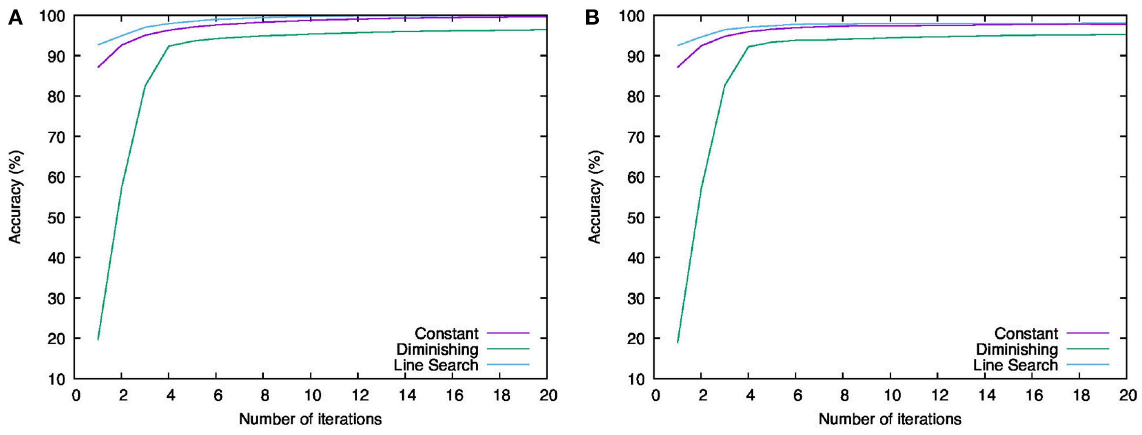

FIGURE 5 | Behavior of the classification accuracies for training and test data. **(A)** Result for training data. **(B)** Result for test data.

the best parameters for the neural network to recognize the handwritten digits.

6. CONCLUSIONS

We proposed novel incremental and parallel subgradient algorithms with a line search that determines suitable learning rates automatically, algorithmically, and appropriately for learning support vector machines. We showed that the algorithms converge to optimal solutions of constrained nonsmooth convex optimization problems appearing in the task of learning support vector machines. Experiments justified the claimed advantages of the proposed algorithms. We compared them with a machine learning algorithm Pegasos, which is designed to learn with a support vector machine efficiently, in terms of prediction accuracy, value of the objective function, and computational time. Regarding the parallel subgradient algorithm in particular, the issue of the computational overhead of the line search can be resolved using multi-core computing. Furthermore, we confirmed that we can apply our incremental subgradient algorithm with the line search to a neural network and they can train it effectively. Overall, our algorithms are useful for efficiently learning a support vector machine and for training a neural network including deep learning.

AUTHOR CONTRIBUTIONS

KH designed the proposed algorithms and analyzed convergence properties of these algorithms and did numerical experiments of them. HI supervised this research all over, designed the proposed algorithms and analyzed convergence properties of these algorithms and evaluated the results of their numerical experiments.

FUNDING

This work was supported by the Japan Society for the Promotion of Science (JSPS KAKENHI Grant Numbers JP17J09220, JP18K11184).

ACKNOWLEDGMENTS

The authors would like to thank the Topic Editor Yoichi Hayashi for giving us a valuable opportunity to submit our research paper to this Research Topic. We are sincerely grateful to the Topic Editor Guido Bologna and the two reviewers for helping us improve the original manuscript. A pre-print version of this paper (Hishinuma and Iiduka, 2016) has been published on the arXiv e-print archive.

REFERENCES

Bauschke, H. H., and Combettes, P. L. (2011). *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. New York, NY: Springer Science+Business Media.

Beltran, C., and Heredia, F. J. (2005). An effective line search for the subgradient method. *J. Optimiz. Theory Appl.* 125, 1–18. doi: 10.1007/s10957-004-1708-4

Berinde, V. (2007). *Iterative Approximation of Fixed Points, Vol. 1912 of Lecture Notes in Mathematics*. Berlin; Heidelberg: Springer–Verlag.

Bertsekas, D. P., Nedić, A., and E., O. A. (2003). *Convex Analysis and Optimization*. Belmont, WA: Athena Scientific.

Bottou, L. (1991). "Stochastic gradient learning in neural networks," in *Proceedings of Neuro-Nîmes 91* (Nimes).

Combettes, P. L. (2003). A block-iterative surrogate constraint splitting method for quadratic signal recovery. *IEEE Trans. Signal Process.* 51, 1771–1782. doi: 10.1109/TSP.2003.812846

Combettes, P. L., and Pesquet, J. C. (2007). A douglas–rachford splitting approach to nonsmooth convex variational signal recovery. *IEEE J. Select. Top. Signal Process.* 1, 564–574. doi: 10.1109/JSTSP.2007.910264

Cristianini, N., and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge: Cambridge University Press.

Cruz, J. Y. B., and Oliveira, W. D. (2016). On weak and strong convergence of the projected gradient method for convex optimization in real Hilbert spaces. *Numer. Funct. Anal. Optimiz.* 37, 129–144. doi: 10.1080/01630563.2015.1080271

Dheeru, D., and Karra Taniskidou, E. (2017). *UCI Machine Learning Repository*. Available online at: https://archive.ics.uci.edu

Hare, W. L., and Lucet, Y. (2014). Derivative-free optimization via proximal point methods. *J. Optimiz. Theory Appl.* 160, 204–220. doi: 10.1007/s10957-013-0354-0

Hayashi, Y., and Iiduka, H. (2018). Optimality and convergence for convex ensemble learning with sparsity and diversity based on fixed point optimization. *Neurocomputing* 273(Supplement. C), 367–372. doi: 10.1016/j.neucom.2017.07.046

Hishinuma, K., and Iiduka, H. (2015). Parallel subgradient method for nonsmooth convex optimization with a simple constraint. *Linear Nonlinear Anal.* 1, 67–77.

Hishinuma, K., and Iiduka, H. (2016). Incremental and parallel line search subgradient methods for constrained nonsmooth convex optimization - numerically accelerated results by multi-core computing. *arXiv:1605.03738 [math.OC]*.

Iiduka, H. (2013). Fixed point optimization algorithms for distributed optimization in networked systems. *SIAM J. Optimiz.* 23, 1–26. doi: 10.1137/120866877

Iiduka, H. (2015a). Acceleration method for convex optimization over the fixed point set of a nonexpansive mapping. *Math. Programm.* 149, 131–165. doi: 10.1007/s10107-013-0741-1

Iiduka, H. (2015b). Parallel computing subgradient method for nonsmooth convex optimization over the intersection of fixed point sets of nonexpansive mappings. *Fixed Point Theory Appl.* 2015:72. doi: 10.1186/s13663-015-0319-0

Iiduka, H. (2016a). Convergence analysis of iterative methods for nonsmooth convex optimization over fixed point sets of quasi-nonexpansive mappings. *Math. Programm.* 159, 509–538. doi: 10.1007/s10107-015-0967-1

Iiduka, H. (2016b). Line search fixed point algorithms based on nonlinear conjugate gradient directions: application to constrained smooth convex optimization. *Fixed Point Theory Appl.* 2016:77. doi: 10.1186/s13663-016-0567-7

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

LeCun, Y., Cortes, C., and Burges, C. J. C. (1998b). *The MNIST Database of Handwritten Digits*. Available online at: http://yann.lecun.com/exdb/mnist/

Leopold, E., and Kindermann, J. (2002). Text categorization with support vector machines. How to represent texts in input space? *Mach. Learn.* 46, 423–444. doi: 10.1023/A:1012491419635

Lin, C.-J. (2017). *LIBSVM Data: Classification, Regression, and Multi-Label.* Available online at: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Lin, Y., Lee, Y., and Wahba, G. (2002). Support vector machines for classification in nonstandard situations. *Mach. Learn.* 46, 191–202. doi: 10.1023/A:1012406528296

Nedić, A., and Bertsekas, D. (2001). *Convergence Rate of Incremental Subgradient Algorithms*.
Boston, MA: Springer US.

Nedić, A., and Bertsekas, D. P. (2001). Incremental subgradient methods for nondifferentiable optimization. *SIAM J. Optimiz.* 12, 109–138. doi: 10.1137/S1052623499362111

Nocedal, J., and Wright, S. (2006). *Numerical Optimization, 2nd Edn*. New York, NY: Springer-Verlag.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* 12, 2825–2830.

Platt, J. (1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Technical report. Microsoft Research.

Polyak, B. T. (1987). *Introduction to Optimization. Translation Series in Mathematics and Engineering.* New York, NY: Optimization Software.

Pradhan, S., Hacioglu, K., Krugler, V., Ward, W., Martin, J. H., and Jurafsky, D. (2005). Support vector learning for semantic argument classification. *Mach. Learn.* 60, 11–39. doi: 10.1007/s10994-005-0912-2

Rockafellar, R. T. (1970). Monotone operators associated with saddle-functions and minimax problems. *Nonlinear Funct. Anal.* 18, 397–407. doi: 10.1090/pspum/018.1/0285942

Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011). Pegasos: primal estimated sub-gradient solver for SVM. *Math. Programm.* 127, 3–30. doi: 10.1007/s10107-010-0420-4

Slavakis, K., and Yamada, I. (2007). Robust wideband beamforming by the hybrid steepest descent method. *IEEE Trans. Signal Process.* 55, 4511–4522. doi: 10.1109/TSP.2007.896252

Takahashi, W. (2009). *Introduction to Nonlinear and Convex Analysis*. Yokohama: Yokohama Publishers, Inc.

Tutsoy, O., and Brown, M. (2016a). An analysis of value function learning with piecewise linear control. *J. Exp. Theor. Artif. Intell.* 28, 529–545. doi: 10.1080/0952813X.2015.1020517

Tutsoy, O., and Brown, M. (2016b). Reinforcement learning analysis for a minimum time balance problem. *Trans. Instit. Measure. Control* 38, 1186–1200. doi: 10.1177/0142331215581638

Wolfe, P. (1969). Convergence conditions for ascent methods. *SIAM Rev.* 11, 226–235. doi: 10.1137/1011036

Yamada, I., and Ogura, N. (2005). Hybrid steepest descent method for variational inequality problem over the fixed point set of certain quasi-nonexpansive mappings. *Numer. Funct. Anal. Optimiz.* 25, 619–655. doi: 10.1081/NFA-200045815

Yuan, G., Meng, Z., and Li, Y. (2016). A modified Hestenes and Stiefel conjugate gradient algorithm for large-scale nonsmooth minimizations and nonlinear equations. *J. Optimiz. Theory Appl.* 168, 129–152. doi: 10.1007/s10957-015-0781-1

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.