Trading Representability for Scalability: Adaptive Multi-Hyperplane Machine for Nonlinear Classification

Zhuang Wang
Knowledge and Decision Systems
Siemens Corporate Research
Princeton, NJ, USA
zhuang.wang@siemens.com

Nemanja Djuric
Department of Computer and Information Sciences
Temple University
Philadelphia, PA, USA
nemanja.djuric@temple.edu

Koby Crammer
Department of Electrical Engineering
The Technion
Haifa, Israel
koby@ee.technion.ac.il

Slobodan Vucetic
Department of Computer and Information Sciences
Temple University
Philadelphia, PA, USA
vucetic@temple.edu

ABSTRACT

Support Vector Machines (SVMs) are among the most popular and successful classification algorithms. Kernel SVMs often reach state-of-the-art accuracies, but suffer from the curse of kernelization due to linear model growth with data size on noisy data. Linear SVMs have the ability to efficiently learn from truly large data, but they are applicable to a limited number of domains due to low representational power. To fill the representability and scalability gap between linear and nonlinear SVMs, we propose the Adaptive Multi-hyperplane Machine (AMM) algorithm that accomplishes fast training and prediction and has capability to solve nonlinear classification problems. AMM model consists of a set of hyperplanes (weights), each assigned to one of the multiple classes, and predicts based on the associated class of the weight that provides the largest prediction. The number of weights is automatically determined through an iterative algorithm based on the stochastic gradient descent algorithm which is guaranteed to converge to a local optimum. Since the generalization bound decreases with the number of weights, a weight pruning mechanism is proposed and analyzed. The experiments on several large data sets show that AMM is nearly as fast during training and prediction as the state-of-the-art linear SVM solver and that it can be orders of magnitude faster than kernel SVM. In accuracy, AMM is somewhere between linear and kernel SVMs. For example, on an OCR task with 8 million highly dimensional training examples, AMM trained in 300 seconds on a single-core processor had 0.54% error rate, which was significantly lower than 2.03% error rate of a linear SVM trained in the same time and comparable to 0.43% error rate of a kernel SVM trained in 2 days on 512 processors. The results indicate that AMM could be an attractive option when solving large-scale classification problems. The software is available at `www.dabi.temple.edu/~vucetic/AMM.html`.

KDD'11, August 21–24, 2011, San Diego, California, USA.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*

General Terms

Algorithms, Performance, Experimentation

Keywords

Large-scale learning, Nonlinear classification, Stochastic gradient descent, Support vector machines

1. INTRODUCTION

In the environment where new large-scale problems are emerging in various disciplines and pervasive computing applications are becoming more common, there is an urgent need for machine learning algorithms that can achieve high accuracy in computationally efficient way. Support Vector Machines (SVMs) [7] provide a powerful paradigm for solving complex classification problems. However, this ability comes at the price of significant computational costs. There has been active research over the last decade to improve efficiency of SVM training [23, 27, 3, 22, 6, 10, 16, 18, 9, 2, 28, 21, 25, 5, 19, 26, 24] and today's state-of-the-art solvers are able to tackle data sets having hundreds of thousands or even millions of highly dimensional examples. Despite the recent advances, there is a fundamental limitation of SVMs with nonlinear kernels that prevents applying them on truly large data sets or data streams. This is a well documented and understood property that on the noisy or highly nonlinear data sets, the size of SVM model, measured in the

number of support vectors, grows linearly with number of training examples [20]. This limitation could explain the recent increased interest in linear SVMs whose model size remains fixed regardless of the data size. However, this comes at a price of significantly reduced representational power of linear SVMs.

As a result, there is a large scalability and representability gap between linear and kernel SVMs. On the kernel SVM side, recent empirical results show that training on data with millions of examples can take days even when using parallel processing techniques and result in a classifier consisting of hundreds of thousands of support vectors. On the linear SVM side, training on such large data takes only seconds on a regular PC and results in a single weight vector as the final classifier. The scalability and short prediction time of linear SVMs make them popular in applications such as text categorization, where examples are sufficiently high dimensional to make the classification problem nearly linearly separable. However, in many other applications, high representational power of kernel SVMs makes them significantly more accurate than linear SVMs. Filling the representability and scalability gap between linear and nonlinear classification is the challenge we intend to address in this paper.

We propose the Adaptive Multi-hyperplane Machine (AMM), a fast learning algorithm applicable to large-scale nonlinear classification problems, which provides a computationally efficient alternative to kernel SVMs. The AMM model consists of a set of weights, each assigned to one of the classes, and it predicts based on the class of weight that provides the largest prediction. The idea of using multiple weights for classification can be traced back to the multi-class SVM [8], which assigns a single weight to each class and predicts the class whose weight gives the largest prediction. By defining the cost function as the regularized margin-based training error, the resulting learning problem becomes convex and can be solved by standard tools of convex optimization. An extension of this idea that allows multiple weights per class has been originally proposed in [1]. Since each weight defines a hyperplane, we refer to this approach as the Multi-hyperplane Machine (MM). There are two important properties of the MM approach. The first is that the resulting optimization problem is nonconvex, such that only convergence to a local optimum can be guaranteed. The second is that it has higher representational power than the single-hyperplane approach. It has been demonstrated experimentally that the benefits of increased representational power outweigh problems with nonconvexity and that MM could accurately solve nonlinear classification problems.

There are two issues that prevent use of MM on large-scale classification problems. The first is that it uses a Sequential Minimal Optimization (SMO) [14] based training algorithm, which is a batch solver that is not suitable for large-scale data. Another is that it requires a user to pre-specify the number of weights per class. Since the optimal number of weights depends on a particular classification problem, this creates a need to select this hyperparameter using an expensive cross-validation procedure. The proposed Adaptive MM approach addresses both of these issues, which makes it practically applicable to very large-scale nonlinear classification problems, as will be illustrated in the experiments.

AMM achieves its computational efficiency through the Stochastic Gradient Descent (SGD), a popular optimization technique that has recently attracted much interest as an alternative to the traditional batch-mode convex optimization for training of SVMs on large-scale data (e.g. [18]). SGD works by sampling labeled examples one at a time and updating the model through (sub-)gradient descent over the instantaneous objective function. Its sequential access to data makes it very suitable for large-scale or online learning. We will show analytically that our implementation of SGD allows AMM to converge to a local optimum, a property shared by the MM.

AMM addresses the problem of selection of number of weights by an adaptive procedure that allows SGD algorithm to automatically select an appropriate number of weights. Specifically, the SGD algorithm starts with a single zero weight assigned to each class and adds new weights as necessary. On simple problems such as linear or near-linear classification, the number of weights remains low, while it can become relatively high on more complex problems. Since the generalization error of AMM grows with the number of weights, as shown by an extension of the generalization theorem from [1], AMM also contains a pruning mechanism that removes small weights, such that it provably results in only a minor degradation in optimization accuracy. In addition to improving generalization error, pruning is also useful because it reduces the cost of training and prediction.

With efficient training and prediction, comparable to linear SVM, and the ability to solve nonlinear classification problems, similar to kernel SVM, AMM fills the scalability and representability gap between linear and nonlinear classification. As such, AMM can be an appealing option when solving large-scale nonlinear classification problems.

Table 1: Summary of notation

$\mathbf{x}$	Instance
y	Label
t, n	Indexes for $\mathbf{x}$
$\mathbf{W}$	Weight matrix
$\mathbf{w}_{i,j}$	j-th weight vector of i-th class (column vector)
i, j, k, z	Indexes for $\mathbf{w}$
$A^{(t)}$	A at the t-th iteration
$P(\mathbf{W})$	Objective function parameterized by $\mathbf{W}$
$P(\mathbf{W}\|\mathbf{z})$	Function of $\mathbf{W}$ given the fixed $\mathbf{z}$
$P^{(t)}(\mathbf{W})$	Instantaneous version of $P(\mathbf{W})$ upon $(\mathbf{x}_t, y_t)$
$\mathbf{W}^*$	The optimal solution of $\min P(\mathbf{W}\|\mathbf{z})$

2. PRELIMINARIES

In this paper we focus on multi-class problems. To facilitate reading, the main notation used in the paper is summarized in Table 1. Let us assume we are given a set of training examples $S = \{(\mathbf{x}_n, y_n), n = 1, ..., N\}$, where instance $\mathbf{x}_n \in \mathbb{R}^D$ is a D-dimensional feature vector and $y_n \in \mathcal{Y} = \{1, ..., M\}$ is the multi-class label. The goal is to learn a function $f : \mathbb{R}^D \to \mathcal{Y}$ that accurately predicts the label of new instances. Next, we will give an overview of multi-class SVMs proposed in [8] and their extension, the Multi-hyperplane Machine (MM) proposed in [1]. This overview will provide the necessary background needed for description of the proposed AMM algorithm, given in Section 3.

2.1 Multi-Class SVM

In multi-class SVM [8], the model $f(\mathbf{x})$ is of the form

$$f(\mathbf{x}) = \text{argmax}_{i \in \mathcal{Y}} g(i, \mathbf{x}), \quad (1)$$

where

$$g(i, \mathbf{x}) = \mathbf{w}_i^T \mathbf{x} \tag{2}$$

is parameterized by the weight vector $\mathbf{w}_i \in \mathbb{R}^D$ of the i-th class. Thus, the predicted label of $\mathbf{x}$ is the class of the weight vector that achieves the maximum value $g(i, \mathbf{x})$. By concatenating all the class-specific weight vectors, we can construct

$$\mathbf{W} = [\mathbf{w}_1 \ \mathbf{w}_2 \ ... \ \mathbf{w}_M]$$

as the $D \times M$ weight matrix representing $f(\mathbf{x})$. Under this setup, the multi-class SVM problem was defined in [8] as

$$\min_{\mathbf{W}} P(\mathbf{W}) \equiv \frac{\lambda}{2}||\mathbf{W}||^2 + \frac{1}{N}\sum_{n=1}^{N} l(\mathbf{W}; (\mathbf{x}_n, y_n)), \tag{3}$$

where $\lambda > 0$ is a regularization parameter that trades off the model complexity defined as the Frobenius norm on $\mathbf{W}$,

$$||\mathbf{W}||^2 = \sum_{i \in \mathcal{Y}} ||\mathbf{w}_i||^2,$$

and the margin-based training loss

$$l(\mathbf{W}; (\mathbf{x}_n, y_n)) = \max\left(0, 1 + \max_{i \in \mathcal{Y} \setminus y_n} g(i, \mathbf{x}_n) - g(y_n, \mathbf{x}_n)\right). \tag{4}$$

It can be seen from (4) that the training loss is zero if the prediction from the correct class is larger by at least one than the maximal prediction from the incorrect classes; otherwise, linear penalty is charged.

2.2 Multi-Hyperplane Machine

In order to increase the expressiveness of the classifier, Aiolli and Sperduti [1] extended the multi-class SVM by allowing multiple weights per class. Let us denote $\mathbf{w}_{i,j}$ as the j-th weight of the i-th class and let us assume that the i-th class has a total of b_i weights. By redefining $g(i, \mathbf{x})$ from (2) as

$$g(i, \mathbf{x}) = \max_j \mathbf{w}_{i,j}^T \mathbf{x}, \tag{5}$$

the classifier $f(\mathbf{x})$ from (1) returns the associated class of the weight with the maximal prediction. With this modification to multi-class SVM, the training loss (4) equals zero if the maximal prediction from the weights of the correct class is by at least one larger than any prediction from weights of the incorrect classes.

We can now concatenate all weights and define

$$\mathbf{W} = \left[\ \mathbf{w}_{1,1}...\mathbf{w}_{1,b_1} \ \middle| \ \mathbf{w}_{2,1}...\mathbf{w}_{2,b_2} \ \middle| \ ... \ \middle| \ \mathbf{w}_{M,1}...\mathbf{w}_{M,b_M} \ \right] \tag{6}$$

where $b_1, ..., b_M$ are the numbers of weights assigned to each of the classes and each block in (6) is a set of class-specific weights. Given this setup, $\mathbf{W}$ is learned by solving the optimization problem (3) where $g(i, \mathbf{x})$ is defined as in (5) and the model complexity is calculated as the Frobenius norm on $\mathbf{W}$ from (6). We call this algorithm the Multi-hyperplane Machine (MM).[1]

We should note that although the MM algorithm allows using different number of weights (i.e. b_i) for different classes, due to practical difficulties in determining these numbers, in [1] all classes are assigned the same number of weights b. This can be suboptimal since, depending on their distribution, different classes might require different numbers of weights.

[1]In [1] this algorithm was called the Multi-Prototype SVM.

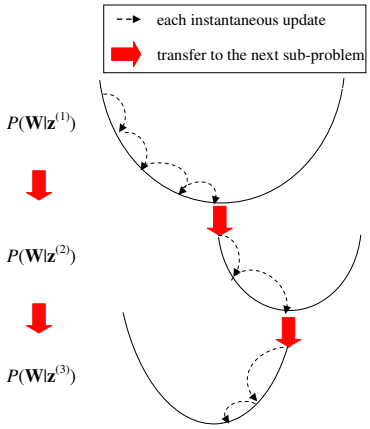

Figure 1: Convergence of MM training

2.3 MM Training

Let us consider finding the optimal MM weight matrix (6) by minimizing (3). As explained in [1], the cost function $P(\mathbf{W})$ is nonconvex and finding the global optimum cannot be guaranteed. To find a local optimum, an iterative procedure was proposed that solves a series of convex upper bounds of $P(\mathbf{W})$. The convex-approximated problem is defined as

$$\min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}) \equiv \frac{\lambda}{2}||\mathbf{W}||^2 + \frac{1}{N}\sum_{n=1}^{N} l_{cvx}(\mathbf{W}; (\mathbf{x}_n, y_n); z_n), \tag{7}$$

where the non-convex loss function l in (3) is replaced by its convex upper bound

$$\begin{aligned} &l_{cvx}(\mathbf{W}; (\mathbf{x}_n, y_n); z_n) \\ &= \max\left(0, 1 + \max_{i \in \mathcal{Y} \setminus y_n} g(i, \mathbf{x}_n) - \mathbf{w}_{y_n, z_n}^T \mathbf{x}_n\right) \end{aligned}, \tag{8}$$

that replaces the concave term $-g(y_n, \mathbf{x}_n)$ in (4) with the convex term $-\mathbf{w}_{y_n,z_n}^T \mathbf{x}_n$. Element z_n of vector $\mathbf{z} = [z_1...z_N]$ determines which weight from the correct class of n-th example is used to calculate (8). Values $\mathbf{z}$ are fixed during optimization of (7).

The resulting MM algorithm can be described in the following way. At step $r = 1$, initialize $\mathbf{z}^{(1)}$, typically by random weight assignment. Then, solve the convex optimization problem (7) to find model weights as

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}^{(r)}).$$

Then, at step $(r + 1)$, recalculate assignments $\mathbf{z}$ such that the objective function is minimized,

$$\mathbf{z}^{(r+1)} = \arg\min_{\mathbf{z}} P(\mathbf{W}^*|\mathbf{z}). \tag{9}$$

This can be achieved by calculating $z_n^{(r+1)}$ as

$$z_n^{(r+1)} = \arg\max_k (\mathbf{w}_{y_n,k}^*)^T \mathbf{x}_n, \tag{10}$$

which is the correct class weight that gives the highest prediction on n-th example. The procedure of optimizing $\mathbf{W}$ and reassigning $\mathbf{z}$ is then repeated until convergence to a local optimum. The convergence is guaranteed because each step of the algorithm leads to a decrease of the objective

function $P(\mathbf{W}|\mathbf{z})$. Convergence of MM is illustrated in Figure 1, where it can be seen that

$$\min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}^{(r)}) \geq \min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}^{(r+1)}).$$

3. ADAPTIVE MULTI-HYPERPLANE MACHINE (AMM)

In this section we describe the AMM algorithm. We start by describing AMM training using Stochastic Gradient Descent (SGD). Then, we explain that SGD allows us to easily address the problem of selecting the number of class-specific weights. After pointing out that the generalization error increases with the number of weights, we propose a pruning strategy that allows reducing the generalization error while having a provably small negative impact on the optimization. Finally, we propose a simplified version of AMM that is suitable for online learning.

3.1 Solving the Sub-Problem (7)

We propose an SGD algorithm to solve the convex optimization problem (7). The SGD is initialized with the zero weight matrix $\mathbf{W}^{(1)} = \mathbf{0}$ and it reads examples one by one and modifies the weight matrix accordingly. Let us for now assume that there are b_i weights for each class. Upon receiving example $(\mathbf{x}_t, y_t) \in S$ at t-th round, $\mathbf{W}^{(t)}$ is updated using the sub-gradient of the instantaneous objective on t-th example defined as

$$P^{(t)}(\mathbf{W}|\mathbf{z}) \equiv \frac{\lambda}{2}||\mathbf{W}||^2 + l_{cvx}(\mathbf{W}; (\mathbf{x}_t, y_t); z_t).$$

As seen, the instantaneous objective differs from (7) in that it only calculates the convex loss on t-th example. The model weights are updated in the negative direction of the sub-gradient as

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta^{(t)} \nabla^{(t)}, \tag{11}$$

where $\eta^{(t)} = 1/(\lambda t)$ is the learning rate and the sub-gradient matrix $\nabla^{(t)}$ is defined as

$$\nabla^{(t)} = \left[\begin{array}{c|c|c|c} \nabla^{(t)}_{1,1} ... \nabla^{(t)}_{1,b_1} & \nabla^{(t)}_{2,1} ... \nabla^{(t)}_{2,b_2} & ... & \nabla^{(t)}_{M,1} ... \nabla^{(t)}_{M,b_M} \end{array} \right] \tag{12}$$

where $\nabla^{(t)}_{i,j} = \nabla_{\mathbf{w}^{(t)}_{i,j}} P^{(t)}(\mathbf{W}^{(t)}|\mathbf{z})$ is a column vector. If convex loss $l_{cvx}(\mathbf{W}; (\mathbf{x}_t, y_t); \mathbf{z}_t)$ equals zero, then $\nabla^{(t)}_{i,j} = \lambda \mathbf{w}^{(t)}_{i,j}$; otherwise,

$$\nabla^{(t)}_{i,j} = \begin{cases} \lambda \mathbf{w}^{(t)}_{i,j} + \mathbf{x}_t, & \text{if} i = i_t,\ j = j_t \\ \lambda \mathbf{w}^{(t)}_{i,j} - \mathbf{x}_t, & \text{if} i = y_t,\ j = z_t \\ \lambda \mathbf{w}^{(t)}_{i,j}, & \text{otherwise,} \end{cases} \tag{13}$$

where

$$i_t = \arg\max_{k \in \mathcal{Y} \setminus y_t} g(k, \mathbf{x}) \quad \text{and} \quad j_t = \arg\max_{k} (\mathbf{w}^{(t)}_{i_t,k})^T \mathbf{x}_t.$$

The update rule (11) can be summarized as follows. At every round, all model weights are reduced towards zero. In addition, if the convex loss on the t-th example is positive, then the class weight from the true class indexed by z_t, $\mathbf{w}^{(t)}_{y_t,z_t}$, is moved towards $\mathbf{x}_t$, while the class weight with the maximum prediction from the remaining classes $\mathbf{w}^{(t)}_{i_t,j_t}$ is moved away from $\mathbf{x}_t$. The SGD updates are summarized in Algorithm 1 (Steps 4-7).

Algorithm 1 Training Algorithm for AMM

Input: Training set S, the regularization parameter λ
Output: $\mathbf{W}^{(t)}$
Initialize: $\mathbf{W}^{(1)} = \mathbf{0}$, $t = 1, r = 1$;
1: initialize $\mathbf{z}^{(1)}$; /* *Build 1-st sub-problem* $P(\boldsymbol{W}|\boldsymbol{z}^{(1)})$ */
2: **repeat**
3: /* *Solve each sub-problem* $P(\boldsymbol{W}|\boldsymbol{z}^{(r)})$*: lines* $4 \sim 7$*/
4: **repeat**
5: $(\mathbf{x}_t, y_t) \leftarrow t$-th example from S;
6: compute $\mathbf{W}^{(++t)}$ using (11);
7: **until** (enough epochs)
8: compute $\mathbf{z}^{(++r)}$ using (9); /* *Reassign* $\boldsymbol{z}$ */
9: **until** ($\mathbf{z}^{(r+1)} == \mathbf{z}^{(r)}$ or enough epochs)

The convergence to the global optimum of the convex subproblem (7) by SGD can be shown by the following theorem. Without the loss of generality, let us assume $||\mathbf{x}|| \leq 1$.

THEOREM 1. *Run the SGD update rule (11) to solve the optimization problem (7). Let* $\boldsymbol{W}^*$ *be the optimal solution of (7). Then we have*

$$\frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{(t)}|\mathbf{z}) - \frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^*|\mathbf{z}) \leq \frac{8(\ln(T)+1)}{\lambda T}.$$

The proof of Theorem 1 is in the Appendix. The theorem tells that when T is large the averaged instantaneous loss of the algorithm converges towards that of the optimal solution. The following corollary can be obtained by following the proof of Theorems 2 and 3 from [18].

COROLLARY 1. *Assume that the conditions stated in Theorem 1 hold and for all* t $(\boldsymbol{x}_t, y_t)$ *is i.i.d. sampled from* S*. Let* $\delta \in (0, 1)$*. Then, with probability of at least* $1 - \delta$*, we have*

$$P(\mathbf{W}^{(T)}|\mathbf{z}) \leq P(\mathbf{W}^*|\mathbf{z}) + \frac{8(\ln(T)+1)}{\delta \lambda T}.$$

The corollary tells that when T is large the weight matrix $\mathbf{W}^{(t)}$ converges to the optimal solution in the limit and that the convergence rate is inversely proportional to the confidence parameter δ.

3.2 Number of Weights

As mentioned in Section 2.2, a drawback of MM is the need to pre-specify the number of model weights. In our AMM algorithm that uses SGD, we address this issue by simply setting the number of weights per class to infinity. Let us discuss the consequences of this idea.

First, we should observe that SGD initializes all model weights to zero. At t-th round, zero weight $\mathbf{w}_{i,j}$ becomes non-zero only if the convex loss $l_{cvx}(\mathbf{W}; (\mathbf{x}_t, y_t); z_t)$ is positive and either (i) $i = i_t$ and $j = j_t$ (the largest prediction from nonzero weights of incorrect classes is less than zero) or (ii) $i = y_t$ and $j = z_t$ (the assigned weight from true class is zero). Therefore, at most two zero weights can become nonzero at each round. In practice, as the AMM classifier improves during training, it will become less likely that any zero weight satisfies either condition (i) or (ii). On more complex and noisy data sets, we can expect to have more nonzero weights than on the simpler classification problems.

Second, let us discuss the implementation issue of storing an infinite number of model weights. We address it by storing all nonzero weights and a single reserved zero weight per class. Any time a zero weight becomes nonzero, we create a space for the newly created nonzero weight. In this way, a compact storage is achieved that is equivalent to storing an infinite number of weights.

3.3 Generalization Error

We now discuss the generalization error of the learned AMM model. Let us denote by b_i the number of AMM weights for i-th class, which is the sum of all its nonzero weights plus one for the reserved zero weight. In [1], the authors derived a generalization error of MM model under the assumption that training data can be separated by the model. Their proof follows the framework of Decision Directed Acyclic Graph (DDAG) [15] and the techniques used in [8]. The proof was based on the assumption that each class has equal number of weights. Here, we provide an extension of the proof in case when the number of weights per class varies.

By considering the classification process of AMM model as a DDAG with $\frac{1}{2}\sum_{i=1}^{M} b_i \sum_{j\neq i}^{M} b_j$ nodes, and following the proof of Lemma 3 in [1], we can obtain a margin-based bound. By upper bounding the margin-relevant terms by the norm of model weights, we have the following error bound.

THEOREM 2. *Suppose we are able to correctly classify an i.i.d sampled training set S using the AMM model*

$$\mathbf{W} = [\mathbf{w}_{1,1}...\mathbf{w}_{1,b_1}\mathbf{w}_{2,1}...\mathbf{w}_{2,b_2}...\mathbf{w}_{M,1}...\mathbf{w}_{M,b_M}],$$

then we can upper bound the generalization error with probability greater than $1-\delta$ as

$$\frac{130}{N}\left(||\mathbf{W}||^2 B\log(4eN)\log(4N) + \log(\frac{2(2N)^K}{\delta})\right),$$

where $B = \sum_{i=1}^{M} b_i + 1 + b_{\max}^2 - b_{\max} - b_{\min}$, $K = \frac{1}{2}\sum_{i=1}^{M} b_i \sum_{j\neq i}^{M} b_j$, $b_{\min} = \min_{i=1,...,M}\{b_i\}$ *and* $b_{\max} = \max_{i=1,...,M}\{b_i\}$.

Theorem 2 shows that the generalization error is proportional to $||\mathbf{W}||^2 B$ and that it can be reduced either by reducing the model weight norm or the number of weights. This clearly suggests that AMM should attempt to use as few nonzero weights as possible. This conclusion is a motivation for weight pruning strategy proposed next.

3.4 Weight Pruning

Every round of SGD training results in shrinking AMM model weights towards zero. As a result, the class weights that are rarely updated can become very small and start resembling zero weights. Such weights are typically generated during initial stages of training when AMM model is less accurate or during model updates caused by noisy examples. Since, by Theorem 2, nonzero weights negatively impact the generalization error, and since replacing small weights with zero weights is not likely to significantly influence AMM accuracy, we propose a pruning step that occasionally removes weights with sufficiently small norms. In addition to reducing the generalization error, weight removal is beneficial from computational viewpoint because it can speed-up both training and prediction and reduce model size.

The pruning step can be formulated as

$$\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t+1)} - \Delta\mathbf{W}^{(t)}, \tag{14}$$

where $\Delta\mathbf{W}^{(t)}$ is a sparse matrix of the same size as $\mathbf{W}^{(t+1)}$ whose nonzero columns correspond to removed weights. For example, if $\mathbf{w}_{i,j}^{(t)}$ is removed, $\Delta\mathbf{w}_{i,j}^{(t)} = \mathbf{w}_{i,j}^{(t)}$, and all other columns of $\Delta\mathbf{W}^{(t)}$ are zero. In AMM, pruning is performed periodically after every k rounds and only on the weights that are below a threshold. The following theorem analyzes the impact of pruning on the convergence of SGD.

THEOREM 3. *Consider the SGD update rule (11). Let $\boldsymbol{W}^*$ be the optimal solution of the problem (7) . Define a pruning constant $c \geq 0$ and execute the pruning step (14) after each k iterations with $\Delta\mathbf{W}^{(t)} \leq c/((t-1)\lambda)$. Then we have*

$$\frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{(t)}|\mathbf{z}) - \frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{*}|\mathbf{z}) \leq \frac{(8+c)(\ln(T)+1)}{\lambda T} + \frac{2(4+c)c}{k\lambda}.$$

The proof is in the Appendix. Theorem 3 quantifies the upper bound on the difference between the averaged instantaneous loss of the AMM with pruning and the optimal solution. It can be seen that the gap is proportional to the pruning threshold c and inversely proportional to the pruning interval k. When $c = 0$, Theorem 3 is equivalent to Theorem 1. Large c and small k correspond to a more aggressive pruning that enforces a simpler classifier but with a wider gap with the optimal solution; while small c and large k shrink the gap but lead to larger AMM model. Next, we state the following important property of pruning.

PROPOSITION 1. *Let us consider the case where pruning (14) is executed after each k iterations with threshold c. Let us consider a weight which is being updated at least c times through the first two cases in (13) during the previous k iterations. It can be shown that such weight will not be pruned.*

We omit the proof due to lack of space. The Proposition 1 provides a very useful interpretation of the effect of the pruning threshold c in Theorem 3. To be consistent with Theorem 3, pruning in AMM is implemented as follows: every k rounds, AMM weights are pruned starting from the one with the smallest norm, and continued as long as the cumulative Frobenius norm of the pruned weights is below threshold $c/((t-1)\lambda)$.

3.5 Online AMM

AMM described in Algorithm 1 is suitable for offline application where the whole data set is residing on a computer and multiple passes through the data are allowed. This is due to the procedure that iteratively solves an approximate convex problem (7) and recalculates assignments $\mathbf{z}$ on the whole data set. Here, we propose a modification of AMM that allows its use in an online setting, where the data set is observed sequentially in a single pass. Online AMM calculates assignment of the incoming t-th example as

$$z_t = \arg\max_k((\mathbf{w}_{y_t,k}^{(t)})^T\mathbf{x}_t)$$

and updates $\mathbf{W}^{(t)}$ to $\mathbf{W}^{(t+1)}$ using SGD (11) with this assignment. The resulting online training procedure is summarized in Algorithm 2.

Algorithm 2 Online Algorithm

Input: Training set S, the regularization parameter λ
Output: $\mathbf{W}^{(t)}$
Initialize: $\mathbf{W}^{(1)} = \mathbf{0}$, $t = 1$

1: **repeat**
2: $(\mathbf{x}_t, y_t) \leftarrow t$-th example from S;
3: calculate $\mathbf{z}_t$ by (10);
4: update $\mathbf{W}^{(++t)}$ by (11);
5: **until** (data set exhausted)

The main difference between the offline (Algorthm 1) and online (Algorithm 2) versions of AMM is that the online version does not wait for convergence of (7) but changes the assignment $\mathbf{z}$ after every update of SGD. As a result, instead of having to periodically calculate assignment $\mathbf{z}$ on the whole data set, it only has to be done on the single incoming example. This allows a single-pass AMM training, and it also leads to savings in computational speed and memory. The price to be paid is that convergence of online AMM to a local optimum cannot be guaranteed, because the online AMM greedily minimizes the non-convex instantaneous objective (3). As will be seen from the experimental results, the online AMM behavior is very similar to the offline AMM. Coupled with computational efficiency, ease of implementation, and applicability for stream learning, this makes online AMM a highly attractive learning algorithm.

3.6 Implementation Details

A naïve implementation of the update rule (11) would require $\mathcal{O}(D)$time for weight shrinking $(1 - \eta^{(t)}\lambda)\mathbf{w}_{i,j}^{(t)}$ and $\mathcal{O}(D)$ time to perform weight update. This computational burden is unnecessary when instances are highly dimensional and sparse (e.g. in text/image data). To circumvent this problem we apply an approach used to speed-up linear SVM training [18] that represents $\mathbf{w}_{i,j}$ as $\mathbf{w}_{i,j} = s_{i,j}\mathbf{v}_{i,j}$, where $s_{i,j}$ is a scalar and $\mathbf{v}_{i,j} \in \mathbb{R}^D$. In this case, the update rule can be decomposed into $s_{i,j}^{(t)} \leftarrow (1 - \eta^{(t)}\lambda)s_{i,j}^{(t)}$ and $\mathbf{v}_{i,j}^{(t)} \leftarrow \mathbf{v}_{i,j}^{(t)} \pm \eta^{(t)}/s_{i,j}^{(t)}\mathbf{x}_t$ in time only proportional to the number of non-zero features in $\mathbf{x}_t$.

4. EXPERIMENTS

In this section, we evaluate the proposed batch and online AMM algorithms and compare them with the large-scale Kernel and Linear SVM algorithms on several large real-life data sets. The competing algorithms are listed as follows:

- **LibSVM**[4] A popular SMO-based SVM solver which is scalable to hundreds of thousands examples.
- **LaSVM**[3] A large-scale online SVM algorithm that accesses one example at a time.
- **P-packSVM**[28] An SVM solver that parallelizes the SGD style training on multiple processors.
- **Pegasos**[18] The state-of-the-art Linear SVM solver which is based on SGD.
- **Poly 2 SVM by Liblinear**[5] A fast solver for Polynomial degree 2 kernel SVM. The algorithm explicitly expresses the feature space as a set of attributes and then trains Liblinear on the transformed data. However, it is only scalable to very sparse or low dimensional data.

A summary of datasets[2] is shown in Table 2.

For Algorithm 1, instead of a random assignment, we initialized $\mathbf{z}^{(1)}$ by a single scan of data using Online AMM, which sequentially sets each element of $\mathbf{z}^{(1)}$ as in (10). Our preliminary results showed it worked better than the simple random initialization approach. We used the epoch-based stopping criteria[3] for SGD for each sub-problem. Since all studied data sets were large we used only 1 epoch before reassigning the weights. We stopped training after 5 passes through the data. We ran online AMM using a single pass of the data. In addition, we also explored performance of online AMM when 5 passes through data. This allowed us to compare online and batch AMM. The pruning step with $k = 10,000$ and $c = 10$ was used with Algorithms 1 & 2 in all the experiments.

The training error and training time of LibSVM, LaSVM, P-packSVM and Poly 2 SVM were taken from recently published results in [5, 12, 28]. For Pegasos and the proposed batch and online AMM algorithms, we selected the regularization parameter λ through cross-validation. The considered range was $\lambda = 10^{-2}, ..., 10^{-7}$. We repeated all the experiments five times, each with randomly shuffled training data. Mean and standard deviation of each set of experiments are reported. All the training examples were normalized such that all attributes were within range $[-1, +1]$. Both batch and online AMM algorithms were implemented in C++. Unless otherwise stated, all the experiments were run using a 3.0GHz Intel Xeon processor with 16G memory on Linux.

The generalization error and the training time of all the algorithms on large datasets are summarized in Table 3. Due to the large computational costs of Kernel SVM training on the largest datasets from our collection, no previous results are reported (to the best of our knowledge), so we use NA to mark these cases in the table. The original MM algorithm [1] is also not scalable to any of the studied large datasets, so it is not included in the comparison.

Batch vs. Online First, let us compare two versions of AMM. We can see that batch AMM achieved lower generalization error than its online sibling at the expense of significantly longer training time that occasionally was more than one order of magnitude longer. The difference in accuracy depends on the difficulty of the datasets. Considering that online AMM used only a single pass of the data, its slightly degraded generalization error is understandable.

AMM vs. Linear SVM Comparing the generalization error of AMM with Linear SVM, we can see that both AMM algorithms significantly outperformed Linear SVM on 5 out of 7 datasets. Considering training time, online AMM had comparable training time to Linear SVM and batch AMM was somewhat slower. Considering prediction time, Table 4 lists the number of weights in the AMM classifiers that dictate prediction time and memory needed to store the

[2]url dataset [13] is available at `http://www.sysnet.ucsd.edu/projects/url/`; all the others are available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

[3]As suggested in [2]. [18] uses a predefined threshold on the objective value as the stopping criteria. However, this threshold can vary from case to case.

Table 2: Summary of large datasets

Datasets	#train	#test	#dim	#class	non-zero %	file size	domain
a9a	32,561	16,281	123	2	11.3%	2M	social survey
ijcnn	49,990	91,701	22	2	66.7%	7.5M	time series
webspam	280,000	70,000	254	2	41.9%	327M	web text
rcv1_bin	677,399	20,242	47,236	2	0.2%	35M	news text
url	1,976,130	420,000	3,231,961	2	0.004%	1.7G	Internet data
mnist8m_bin	8,000,000	10,000	784	2	19.3%	18G	OCR images
mnist8m_mc	8,000,000	10,000	784	10	19.3%	18G	OCR images

Table 3: Error rate and training time comparison with large-scale algorithms (RBF SVM is solved by LibSVM unless specified otherwise. Poly2 and LibSVM results are from [5]).

Datasets	Error rate (%)					Training time (seconds)[1]				
	AMM *batch*	AMM *online*	Linear (Pegasos)	Poly2 SVM	RBF SVM	AMM *batch*	AMM *online*	Linear (Pegasos)	Poly2 SVM	RBF SVM
a9a	15.03±0.11	16.44±0.23	15.04±0.07	14.94	14.97	2	0.2	1	2	99
ijcnn	2.40±0.11	3.02±0.14	7.76±0.19	2.16	1.31	2	0.1	1	11	27
webspam	4.50±0.24	6.14±1.08	7.28±0.09	1.56	0.80	80	4	12	3,228	15,571
mnist_bin	0.53±0.05	0.54±0.03	2.03±0.04	NA	0.43[2]	3084	300	277	NA	2 days[2]
mnist_mc	3.20±0.16	3.36±0.20	8.41±0.11	NA	0.67[3]	13864	1200	1180	NA	8 days[3]
rcv1_bin	2.20±0.01	2.21±0.02	2.29±0.01	NA	NA	1100	80	25	NA	NA
url	1.34±0.21	2.87±1.49	1.50±0.39	NA	NA	400	24	100	NA	NA

[1] excludes data loading time.
[2] achieved by parallel training P-packSVMs on 512 processors; results from [28].
[3] achieved by LaSVM; results from [12].

classification model. These numbers divided by the number of classes directly reflect the increased prediction time of AMM as compared to Linear SVM. We can see that AMM classifiers are around one order of magnitude slower than Linear SVM, which has $\mathcal{O}(MD)$ prediction time. This is very impressive result considering that AMM achieves significantly lower error rate than Linear SVM. This indicates that AMM is superior to linear SVM because it provides a better tradeoff between accuracy and speed in all but the most resource-constrained applications.

AMM vs. Kernel SVM Comparison against Poly2 SVM on three relatively small datasets (a9a, ijcnn, webspam) shows that AMM had similar error rates as Poly2 SVM, while achieving several orders of magnitude faster training. Also, the $\mathcal{O}(MD)$ prediction time of AMM is much more favorable than the $\mathcal{O}(MD^2)$ prediction time of Poly2 SVM. By comparing AMM with RBF SVM on 5 low-dimensional datasets (where RBF SVM's results were reported in the previous literature), we can see that AMM has somewhat lower ($0.1\% - 2.9\%$) but still comparable error rate. Particularly, on the two largest datasets (mnist8m_bin and mnnist8m_mul), the error rate achieved by AMM after running on a single processor for a couple of hours is very competitive to P-packSVM's classifier trained in 2 days on 512 processors and LaSVM's classifier trained in 8 days on a single processor. These results show that the AMM algorithms are appealing alternative to kernel SVMs when learning from very large data.

Train Online AMM in Batch Mode Table 3 lists Online AMM results when it was run with a single pass through the data. We also performed experiments when it was allowed to make multiple passes through the data. The detailed results on the two largest datasets for the multi-pass Onine AMM are shown in Figure 2. The top left and bottom left panels in Figure 2 show the evolution of the error rate and the total training time as a function of the number of epochs (one epoch denotes a full pass through the data). We can see that the error rate rapidly decreases during the first epoch and that it continues to improve slightly after the first epoch. The training time increases linearly, as expected. The results suggests that, if the training time is not of major concern and if data could be stored in memory, multiple accesses to the training data should be recommended. The top right and bottom right panels of Figure 2 show the number of AMM weights as training progresses. In addition to the current number of weights, the panels also show the total number of weights created and pruned at any stage of training process. It can be seen that the pruning has the desired effect of controlling the total number of model weights without negatively influencing the error rate.

5. CONCLUSIONS

Recent advances in large-scale learning resulted in many successful algorithms for linear classification. However, creating sufficiently efficient kernel SVM is still an open problem that prevents its application on the largest data sets. This study aimed at filling the scalability and representability gap between linear and kernel SVMs. We presented two multi-hyperplane algorithms for multi-class classification that have several favorable features: fast training and prediction, simple implementation, ability to represent non-linear concepts, and theoretical justification of their proper-

Table 4: The number of weights in the classifiers

Datasets	a9a	ijcnn	webspam	mnist8m_bin	mnist8m_mc	rcv1	url
batch AMM	11±1	11±1	13±2	20±1	65±2	22±2	4±0
online AMM	16±2	15±1	10±1	13±1	61±3	44±5	5±1

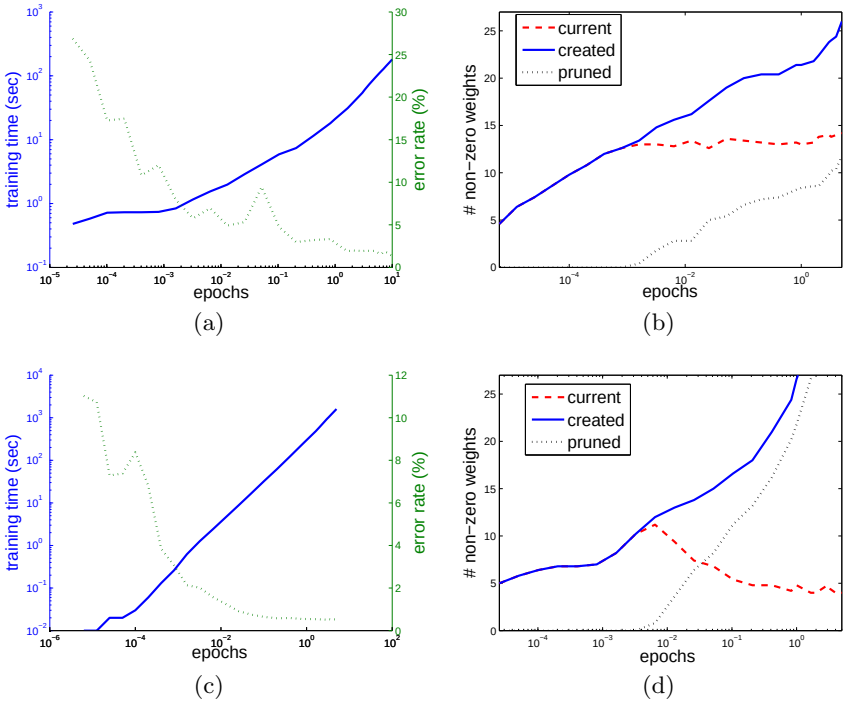

Figure 2: Detailed results on url data (upper two panels) and mnist8m_bin data (lower two panels)

ties. Experimental comparison with the state-of-the-art linear and kernel SVMs on large data sets showed that AMM classifiers provide an interesting tradeoff between these two major classes of algorithms. In addition to being an attractive option when solving large-scale classification problems, AMM could also be useful as a data exploration tool. When faced with a new large-scale classification problem, AMM could be trained together with linear SVM and if its error rate is significantly lower, it could indicate that the problem is nonlinear and that more computationally costly but more powerful classification algorithms should be considered.

6. ACKNOWLEDGMENTS

Authors thank Yu Liang (Temple Univ.) for implementing the first version of the algorithms and Kai Zhang (Siemens Corporate Research) for proofreading the manuscript. N. Djuric and S. Vucetic were supported by the U.S. National Science Foundation Grant IIS-0546155. K. Crammer is a Horev Fellow, supported by the Taub Foundations.

7. REFERENCES

[1] F. Aiolli and A. Sperduti. Multi-class classification with multi-prototype support vector machines. *Journal of Machine Learning Research*, 2005.

[2] A. Bordes, L. Bottou, and P. Gallinari. Sgd-qn: careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 2009.

[3] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers for online and active learning. *Journal of Machine Learning Research*, 2005.

[4] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines, http://www.csie.ntu.edu.tw/ cjlin/libsvm. 2001.

[5] Y.-W. Chang, K.-W. C. C.-J. Hsie and, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 2010.

[6] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *International Conference on Machine Learning*, 2006.

[7] Cortes and Vapnik. Support-vector networks. *Machine Learning*, 1995.

[8] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2001.

[9] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *International Conference on Machine Learning*, 2008.

[10] T. Joachims. Training linear svms in linear time. In

ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2006.

[11] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 2002.

[12] G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. *Large Scale Kernel Machines, Cam-bridge, MA, MIT Press*, 2007.

[13] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *International Conference on Machine Learning*, 2009.

[14] J. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods - support vector learning, MIT Press*, 1998.

[15] J. Platt, N. Cristianini, and J. S. Taylor. Large margin dags for multiclass classification. In *Advance in Nueral Information Processing Systems*, 2000.

[16] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advance in Nueral Information Processing Systems*, 2007.

[17] S. Shalev-Shwartz and Y. Singer. Logarithmic regret algorithms for strongly convex repeated games (technical report). The Hebrew University, 2007.

[18] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: primal estimated sub-gradient solver for svm. In *International Conference on Machine Learning*, 2007.

[19] S. Sonnenburg and V. Franc. Coffin : a computational framework for linear svms. In *International Conference on Machine Learning*, 2010.

[20] I. Steinwart. Sparseness of support vector machines. *Journal of Machine Learning Research*, 2003.

[21] C. Teo, S. V. N. Vishwanathan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 2010.

[22] I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core vector machines: fast svm training on very large data sets. *Journal of Machine Learning Research*, 2005.

[23] S. V. N. Vishwanathan, A. J. Smola, and M. N.Murty. Simplesvm. In *International Conference on Machine Learning*, 2003.

[24] Z. Wang, K. Crammer, and S.Vucetic. Multi-class pegasos on a budget. In *International Conference on Machine Learning*, 2010.

[25] Z. Wang and S.Vucetic. Online training on a budget of support vector machines using twin prototypes. *Statisitcal Analysis and Data Mining Journal*, 2010.

[26] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2010.

[27] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent. In *International Conference on Machine Learning*, 2004.

[28] Z. A. Zhu, W. Chen, G. Wang, C. Zhu, and Z. Chen. P-packsvm: parallel primal gradient descent kernel svm. In *IEEE International Conference on Data Mining*, 2009.

APPENDIX

Unified Proof of Theorems 1 & 3: First, we rewrite the update rule of SGD with the optional pruning step as $\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t)} - \eta^{(t)}\partial^{(t)}$, where $\partial^{(t)} = \nabla^{(t)} + E^{(t)}$ and $E^{(t)} = \mathbf{0}$ if no pruning is used. The relative progress towards the optimal solution $\mathbf{W}^*$ at t-th round $D^{(t)}$ can be lower bounded as

$$\begin{aligned}
&D^{(t)} = ||\mathbf{W}^{(t)} - \mathbf{W}^*||^2 - ||\mathbf{W}^{(t)} - \eta^{(t)}\nabla^{(t)} - \eta^{(t)}E^{(t)} - \mathbf{W}^*||^2 \\
&= -(\eta^{(t)})^2||\partial^{(t)}||^2 + 2\eta^{(t)}(E^{(t)})^T(\mathbf{W}^{(t)} - \mathbf{W}^*) \\
&+2\eta^{(t)}(\nabla^{(t)})^T(\mathbf{W}^{(t)} - \mathbf{W}^*) \\
&\geq_1 -(\eta^{(t)})^2G^2 - 2\eta^{(t)}||E^{(t)}||\tfrac{4+c}{\lambda} \\
&+2\eta^{(t)}\left(P^{(t)}(\mathbf{W}^{(t)}) - P^{(t)}(\mathbf{W}^*) + \tfrac{\lambda}{2}||\mathbf{W}^{(t)} - \mathbf{W}^*||^2\right).
\end{aligned} \tag{15}$$

In $\geq_1$, we assume there is a constant $G \geq 0$ such that $||\partial^{(t)}|| \leq G$ and we will quantify G later on. For the second term, we first bound $\mathbf{W}^{(t)}$ as

$$\begin{aligned}
&||\mathbf{W}^{(t)}|| \leq ||(1 - \eta^{(t-1)}\lambda)\mathbf{W}^{(t-1)}|| + 2\eta^{(t-1)} + ||\Delta\mathbf{W}^{(t-1)}|| \\
&\leq \tfrac{t-2}{t-1}||\mathbf{W}^{(t-1)}|| + \tfrac{2}{(t-1)\lambda} + \tfrac{c}{(t-1)\lambda} \\
&\leq \tfrac{1}{t-1}||\mathbf{W}^{(0)}|| + \tfrac{2(t-1)}{(t-1)\lambda} + \tfrac{(t-1)c}{(t-1)\lambda} = \tfrac{2+c}{\lambda},
\end{aligned}$$

and then use triangle inequality to bound $||\mathbf{W}^{(t)} - \mathbf{W}^*|| \leq (2 + c)/\lambda + 2/\lambda$ by using the fact $||\mathbf{W}^*|| \leq 2/\lambda$ according to the result in [11]. Then, we bound the third term using function $P^{(t)}(\mathbf{W}^{(t)})$'s λ-strong convexity [17].

Dividing both sides of inequality (15) by $2\eta^{(t)}$ and rearranging, we obtain

$$\begin{aligned}
&P^{(t)}(\mathbf{W}^{(t)}) - P^{(t)}(\mathbf{W}^*) \leq \tfrac{D^{(t)}}{2\eta^{(t)}} - \tfrac{\lambda}{2}||\mathbf{W}^{(t)} - \mathbf{W}^*||^2 \\
&+\tfrac{\eta^{(t)}G^2}{2} + \tfrac{(4+c)||E^{(t)}||}{\lambda}.
\end{aligned} \tag{16}$$

Summing over all t, we get

$$\begin{aligned}
&\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{(t)}) - \sum_{t=1}^{T} P^{(t)}(\mathbf{W}^*) \leq \sum_{t=1}^{T} \tfrac{D^{(t)}}{2\eta^{(t)}} \\
&- \sum_{t=1}^{T} \tfrac{\lambda}{2}||\mathbf{W}^{(t)} - \mathbf{W}^*||^2 + \tfrac{G^2}{2}\sum_{t=1}^{T}\eta^{(t)} + \tfrac{(4+c)}{\lambda}\sum_{t=1}^{T}||E^{(t)}||.
\end{aligned}$$

We bound the first and second terms in inequality (16) as

$$\begin{aligned}
&\tfrac{1}{2}\sum_{t=1}^{N}\left(\tfrac{D^{(t)}}{\eta^{(t)}} - \lambda||\mathbf{W}^{(t)} - \mathbf{W}^*||^2\right) = \tfrac{1}{2}\Big((\tfrac{1}{\eta^{(1)}} - \lambda)||\mathbf{W}^{(1)} \\
&-\mathbf{W}^*||^2 + \sum_{t=2}^{N}\left(\tfrac{1}{\eta^{(t)}} - \tfrac{1}{\eta^{(t-1)}} - \lambda\right)||\mathbf{W}^{(t)} - \mathbf{W}^*||^2 \\
&-\tfrac{1}{\eta^{(N)}}||\mathbf{W}^{(T+1)} - \mathbf{W}^*||^2\Big) \\
&=_1 -\tfrac{1}{2\eta^{(N)}}||\mathbf{W}^{(T+1)} - \mathbf{W}^*||^2 \leq 0.
\end{aligned} \tag{17}$$

In $=_1$, the first and second terms vanish after plugging $\eta_t \equiv 1/(\lambda t)$. Next, we bound the third term in inequality (16) according to the divergence rate of harmonic series,

$$\frac{G^2}{2}\sum_{t=1}^{T}\eta^{(t)} = \frac{G^2}{2\lambda}\sum_{t=1}^{T}\frac{1}{t} \leq \frac{G^2}{2\lambda}(\ln(T) + 1). \tag{18}$$

Then, we quantify G as

$$||\partial^{(t)}|| \leq ||\nabla^{(t)}|| + ||E^{(t)}|| \leq (\lambda||\mathbf{W}^{(t)}|| + 2) + 2 + c \leq 6 + c \equiv G.$$

If no pruning is used, $||E^{(t)}|| = 0$. For the iterations when the pruning is executed, we bound $||E^{(t)}||$ as $||E^{(t)}|| \leq 2c$ using the bound on $\Delta\mathbf{W}^{(t)}$. Combining inequality (17) with (18) and dividing two sides of inequality by T, we get the stated bounds as in Theorems 1 and 3.