Parallel and Distributed Structured SVM Training

Jiantong Jiang*†, Zeyi Wen*[1], Zeke Wang†, Bingsheng He‡ and Jian Chen§
†Zhejiang University, *The University of Western Australia,
‡National University of Singapore, §South China University of Technology
jiantong.jiang@research.uwa.edu.au, zeyi.wen@uwa.edu.au, wangzeke@zju.edu.cn,
hebs@comp.nus.edu.sg, ellachen@scut.edu.cn

Abstract—Structured Support Vector Machines (structured SVMs) are a fundamental machine learning algorithm, and have solid theoretical foundation and high effectiveness in applications such as natural language parsing and computer vision. However, training structured SVMs is very time-consuming, due to the large number of constraints and inferior convergence rates, especially for large training data sets. The high cost of training structured SVMs has hindered its adoption to new applications. In this paper, we aim to improve the efficiency of structured SVMs by proposing a parallel and distributed solution (namely *FastSSVM*) for training structured SVMs building on top of MPI and OpenMP. FastSSVM exploits a series of optimizations (e.g., optimizations on data storage and synchronization) to efficiently use the resources of the nodes in a cluster and the cores of the nodes. Moreover, FastSSVM tackles the large constraint set problem by batch processing and addresses the slow convergence challenge by adapting stop conditions based on the improvement of each iteration. We theoretically prove that our solution is guaranteed to converge to a global optimum. A comprehensive experimental study shows that FastSSVM can achieve at least four times speedup over the existing solutions, and in some cases can achieve two to three orders of magnitude speedup.

Index Terms—Parallel and Distributed Training, Structured Machine Learning, Support Vector Machines.

1 Introduction

Although deep learning, particularly Deep Neural Network (DNN) based techniques, has been offering state-of-the-art solutions for many problems [1], [2], [3], [4], DNN based techniques have apparent deficiencies including the need of huge amount of data and hyper-parameters. Even worse, theoretical analysis of DNN based techniques is difficult due to too many factors with almost infinite number of hyper-parameter configuration combinations. In comparison, structured Support Vector Machines (structured SVMs) have excellent theoretical foundation [5], and they are much simpler to use due to much fewer hyper-parameters to be tuned. Structured SVMs have been providing an important approach to deal with complex multi-dimensional structured prediction problems, which have shown high effectiveness in structured prediction problems [6], [7]. In fact, research work has been using structured SVMs to solve their problems from natural language parsing [8], [9], computer vision [10], [11], [12], [13] and biomedical engineering [14], [15], [16]. These problems cannot be solved by ordinary SVMs which do not support structured outputs.

However, a key barrier that hinders the wider usage of structured SVMs is its long training time especially for large and complex problems. We have conducted experiments on publicly available data sets with the existing implementations SVM-Struct from SVM^{light} [6]. The experimental results shown in Figure 1 indicate that the training of SVM-Struct is indeed time-consuming (e.g., 10 hours of training on the *pendigits* data set). By using our proposed techniques in this work, the training time can be dramatically reduced (e.g., the training only needs 4 minutes in *pendigits*).

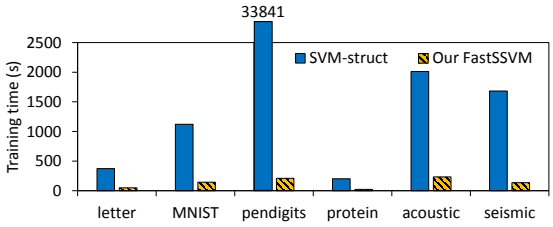


Fig. 1: Training time of SVM-Struct and FastSSVM for multi-class classification tasks on publicly available data sets.

While much research has been conducted on the acceleration of ordinary SVMs (e.g., ThunderSVM [17]), few research work has been dedicated to accelerating structured SVMs. This work aims to fill the gap by exploiting parallel and distributed processing to accelerate the training procedure of structured SVMs. Nevertheless, training structured SVMs efficiently is challenging due to the following reasons. First, the training procedure is inherently sequential due to dependencies between optimization steps, hence parallelization is not trivial. Second, compared with ordinary SVMs, structured SVMs bear a great burden of communication due to the relatively large size of constraint information to be transferred. Third, structured SVM training procedure may suffer inferior convergence rates, especially for complex problems with a large data set [18].

To overcome the challenges, we present *FastSSVM*, a parallel and distributed structured SVM training solution with Message Passing Interface (MPI) and OpenMP, with inter-node and intra-node optimizations. In the distributed multi-node system level (i.e., inter-node level), we develop a cascade architecture, where smaller optimizations are solved

[1] *Zeyi Wen is the corresponding author.*

(a) object detection (b) object segmentation (c) syntactic parsing

Fig. 2: Example tasks for structured SVM learning: (a) and (b) are from PASCAL VOC dataset [20], and (c) is from WSJ Corpus [21].

independently and the partial results are combined in a hierarchical fashion to guarantee convergence. We exploit centralized and local storage of data for efficient communication given the available resources. In the single node level (i.e., intra-node level), we develop techniques to efficiently synchronize the intermediate weights to take advantage of the computing power of multi-cores. Moreover, we (i) alleviate the problem of a large number of constraints by handling the constraints in batches and (ii) mitigate the slow convergence rate by adapting stop conditions being aware of the improvement gained by each training iteration. In summary, our key contributions in this paper are as follows.

- We propose a solution for training structured SVMs with a cascade architecture to split the data set and to optimize the subsets independently. We theoretically prove that our solution is guaranteed to converge to a global optimum.
- To improve the efficiency, we design a hybrid MPI+OpenMP solution, namely FastSSVM, for training structured SVMs in parallel and distributed environment. To relieve the communication burden and achieve high efficiency, we develop a series of novel techniques, including centralized and local storage of training data and synchronization of intermediate weights by temporary separation and sharing.
- We propose mechanisms to alleviate inferior convergence rates for the problems with many constraints, so that FastSSVM can scale to large scale cluster systems.
- We conduct extensive experiments to study the impact of different optimizations and the efficiency of FastSSVM. Experimental results show that FastSSVM outperforms the existing solution SVM-Struct [6] and FSMO [19] by at least four times, and can achieve three orders of magnitude speedup for some problems, which demonstrates the effectiveness of our proposed techniques.

We plan to integrate *FastSSVM* into ThunderSVM [17] such that users are able to easily and efficiently leverage the computing power of a computer cluster when using *FastSSVM* to solve their problems.

2 Preliminaries

In this section, we first present details of structured SVMs. Then we review the cutting-plane [22] algorithm for training structured SVMs and fixed-threshold sequential minimal optimization (FSMO) [19] algorithm for solving quadratic programming (QP) optimization problems.

2.1 Structured SVMs

Structured SVMs [6], [7] are commonly used for structured learning problems. An input $x_i \in X$ is attached with an output $y_i \in Y$, where elements in Y are structured objects such as sequences, strings, trees, or bounding boxes. Given a training set in the input space X and output space Y, the goal of structured SVM training is to learn a discriminant function that minimizes the empirical risk. Example tasks are common in computer vision, natural language processing, and many other application domains, as shown in Figure 2. For instance, for object detection and object segmentation structured prediction tasks, the input are both the images; after the inference procedure of Structured SVMs, we can get the structured output that are the bounding boxes and segmentation masks, respectively.

Formally, the structured SVM problem can be expressed as a quadratic programming (QP) optimization problem with many constraints, where a margin re-scaling approach is proposed for the case of arbitrary loss functions [7].

$$\min_{\boldsymbol{w},\xi} \frac{1}{2}||\boldsymbol{w}||^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \tag{1}$$
$$\text{s.t.} \quad \forall i, \xi_i \geq 0, \quad \forall i, \forall \bar{y} \in Y \backslash y_i : \boldsymbol{w} \cdot \delta\Psi_i(x_i, \bar{y}) \geq \Delta(y_i, \bar{y}) - \xi_i$$

where $\boldsymbol{w}$ is the weight vector, ξ_i is the slack variable to tolerate misclassification on training instance x_i, C is the regularization constant. For ease of presentation, we define $\delta\Psi_i(x_i, \bar{y}) \equiv \Psi(x_i, y_i) - \Psi(x_i, \bar{y})$ in the above optimization problem. $\Psi(x, y)$ is the feature vector extracted from both the input x and output y, which depends on the nature of the task and the goal is to learn a function $F_w(x, y) = \boldsymbol{w} \cdot \delta\Psi(x, y)$. Intuitively, one can think of $F_w(x, y)$ as a compatibility function that measures how relevant x and y are. The objective function in (1) is the conventional regularized risk used in traditional SVMs. The constraints state that for each training instance (x_i, y_i), the score $F_w(x_i, y_i)$ of the correct structure y_i must be greater than or equal to the score $F_w(x_i, \bar{y})$ of all incorrect structures $\bar{y}$ by a margin $\Delta(y_i, \bar{y})$ minus a slack ξ_i. This optimization problem is referred to as the primal problem in the convex optimization research community [23]. For a new input x_p, the discriminant function $F_w(x, y)$ learned in the above optimization problem can be used to predict the structured output y_p, which is $y_p = argmax_{\bar{y} \in Y} F_w(x_p, \bar{y})$.

2.1.1 Working Set Selection

The optimization problem (1) has a large (typically exponential or infinite) number of constraints. The number of constraints equals the number of all possible $\bar{y} \in Y \backslash y_i$ for all instances of the training set, which is task dependent. For example, for multi-class classification task, $\#constraint = (\#class - 1) \times \#instance$, while for multi-label classification task, $\#constraint = (2^{\#class} - 1) \times \#instance$, which grows at an exponential rate. Therefore, solving the problem by considering all the constraints at once is extremely expensive. One idea to efficient solve the problem is to select some active constraints to construct the working set to build an approximation to the whole constraint set. Then the subproblem can be solved over the working set and ensures sufficient accuracy. Joachimes et al. proposed a cutting-plane

Algorithm 1: The cutting-plane algorithm

1 **Input:** $\{(x_i, y_i)|i = 1, 2, \cdots, n\}$, C, ε
2 $\mathcal{S}_i \leftarrow \emptyset, \xi_i \leftarrow 0$ for all $i = 1, \cdots, n$
3 **repeat**
4 **for** $i = 1, \cdots, n$ **do**
5 $\bar{y} \leftarrow argmax_{\bar{y} \in \mathcal{Y}} \{\Delta(y_i, \bar{y}) - \boldsymbol{w} \cdot \delta\Psi_i(x_i, \bar{y})\}$
6 $\xi_i \leftarrow max_{\bar{y} \in \mathcal{S}_i} \{\Delta(y_i, \bar{y}) - \boldsymbol{w} \cdot \delta\Psi_i(x_i, \bar{y})\}$
7 **if** $\Delta(y_i, \bar{y}) - \boldsymbol{w} \cdot \delta\Psi_i(x_i, \bar{y}) > \xi_i + \varepsilon$ **then**
8 $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{\bar{y}\}$
9 $\alpha_{\mathcal{S}} \leftarrow$ solve the sub-problem over $\mathcal{S} = \bigcup_i \mathcal{S}_i$
10 **end if**
11 **end for**
12 **until** no $\mathcal{S}_i$ has changed during iteration

Algorithm 2: FSMO

1 **Input:** $\{(x_i, y_i)|i = 1, 2, \cdots, n\}$, $\mathcal{S}$, $\boldsymbol{\alpha_{\mathcal{S}}}$, C
2 $\boldsymbol{w} = \sum_{i, \bar{y} \neq y_i} \alpha_{i\bar{y}} \delta\Psi_i(x_i, \bar{y})$
3 **repeat**
4 **for** $(x_i, \bar{y})$ in $\mathcal{S}$ **do**
5 **if** $(x_i, \bar{y})$ violates the KKT condition **do**
6 $\alpha_{i\bar{y}} \leftarrow \alpha_{i\bar{y}}^{old} + \frac{\Delta(y_i, \bar{y}) - \boldsymbol{w} \cdot \delta\Psi_i(x_i, \bar{y})}{\|\delta\Psi_i(x_i, \bar{y})\|^2}$, $\sum_{\bar{y} \neq y_i} \in [0, \frac{C}{n}]$
7 $\boldsymbol{w} \leftarrow \boldsymbol{w}^{old} + (\alpha_{i\bar{y}} - \alpha_{i\bar{y}}^{old}) \delta\Psi_i(x_i, \bar{y})$
8 **end if**
9 **end for**
10 **until** no $\alpha_{i\bar{y}}$ changed during iteration

method [22] which gradually adds constraints during the training. The key steps of the algorithm is summarized in Algorithm 1. It maintains a working set $\mathcal{S}$ and proceeds by iteratively alternating between i) finding the most violated constraint to add to the working set (Line 5 and Line 8) and ii) recomputing the solution of the sub-problem over the current working set once a constraint has been added (Line 9). The algorithm terminates when all the constraints satisfy the optimality condition (Line 12).

2.1.2 The Fixed-Threshold Sequential Minimal Optimization (FSMO) Algorithm

For structured SVM learning, one important thing is to recognize a set of support vectors from working set. Solving the QP problem is to identify all the support vectors. The common and more intuitive approach for finding the support vectors is to represent the primal problem (1) in a dual form [23]. The QP problem represented in the dual form is shown in problem (2) below.

$$\begin{aligned} \min_{\alpha} L(\alpha) = & \frac{1}{2} \sum_{i, \bar{y} \neq y_i} \sum_{j, \bar{y} \neq y_j} \alpha_{i,\bar{y}} \alpha_{j,\bar{y}} \delta\Psi_i(x_i, \bar{y}) \cdot \delta\Psi_j(x_j, \bar{y}) \\ & - \sum_{i, \bar{y} \neq y_i} \alpha_{i,\bar{y}} \Delta(y_i, \bar{y}) \\ \text{s.t.} \quad & \forall i, 0 \leq \sum_{i, \bar{y} \neq y_i} \alpha_{i,\bar{y}} \leq \frac{C}{n}, \qquad \forall i, \bar{y}, \alpha_{i,\bar{y}} \geq 0 \end{aligned} \tag{2}$$

The objective function is solely dependent on a set of α. Once the αs are obtained, the other primal variables can be easily determined. In analogy to traditional SVMs, we refer to those predicted $\bar{y}$ with non-zero $\alpha_{i,\bar{y}}$ as support vectors. The set of α is called Lagrangian multipliers.

The dual QP problem can be solved by the fixed-threshold sequential minimal optimization (FSMO) algorithm which is simple and faster than the standard SVM training algorithms for structured SVM problems as shown by [19]. The key idea is to adjust the value of one α at a time until all the αs satisfy the optimality condition. The pseudocode of FSMO is given in Algorithm 2. By traversing all the training instances $(x_i, \bar{y})$ in working set $\mathcal{S}$, the algorithm finds the constraint which violates the KKT conditions [24] shown in Equation 3, and iteratively updates $\alpha_{i,\bar{y}}$ and w.

In this work, we focus on accelerating FSMO algorithm for solving the QP problem.

$$\begin{aligned} & \Delta(y_i, \bar{y}) - \boldsymbol{w} \cdot \delta\Psi_i(x_i, \bar{y}) = 0, \quad \text{if} \quad 0 < \alpha_{i\bar{y}} < \frac{C}{n} \\ & \Delta(y_i, \bar{y}) - \boldsymbol{w} \cdot \delta\Psi_i(x_i, \bar{y}) > 0, \quad \text{if} \quad \alpha_{i\bar{y}} = \frac{C}{n} \\ & \Delta(y_i, \bar{y}) - \boldsymbol{w} \cdot \delta\Psi_i(x_i, \bar{y}) < 0, \quad \text{if} \quad \alpha_{i\bar{y}} = 0 \end{aligned} \tag{3}$$

3 Optimization for Better Parallelism

We find two major issues in the existing implementations. First, the number of constraints is large, it is inefficient to only add one constrant to the working set at each iteration. Second, the learning procedure may result in inferior convergence rates, especially for problems with many parameters and constraints. In this section, we aim to optimize the learning procedure to tackle these two major issues.

3.1 Handling Constraints in Batches

The cutting-plane method builds an approximation to the constraint set by adding the most violated constraint in each iteration. However, the naive approach that solving the sub-QP problem once a constraint is added is inefficient. To speed up the cutting-plane algorithm, our idea is to efficiently build a richer approximation in each iteration. Instead of adding one constraint in each iteration, we propose to obtain multiple violated constraints in batches before solving the sub-QP problem.

To achieve faster convergence, each selected constraint should violate the optimality condition as much as possible, while the batch of constraints should be diverse (i.e. the sub-problem is a good representation of the whole problem). For instance, if we just add the same constraint multiple times, the duplicated constraints does not contribute to the convergence rate. Therefore, we scan all instances within each epoch and find only one most violated constraint of each instance to ensure that the constraints being added at the same iteration belong to different training instances. Meanwhile, we find that the training time can be reduced with the increase of batch size in practice. However, if the batch size is too large, the accuracy is impaired. Since the total number of constraints is proportional to the number of training instances in Structured SVM training, we set the batch size be proportional to the number of instances, i.e., $batch\_size = \#instances/100$.

3.2 Training Convergence Improvement

In order to alleviate the problem of slow convergence or non-convergence, we add stop conditions according to the change of the dual value in two levels, including the inner FSMO solver and outer cutting plane training.

Inner FSMO Solver. When FSMO has enough iterations (e.g., 2000 times), we start to check the dual values between two adjacent iterations. If the change of dual value is less than a certain threshold (e.g., 0.1) for several consecutive times (e.g., 5 times), the iteration of FSMO can be stopped in advance, even if some $\alpha_{i,\bar{y}}$ can continue to be updated. We employ the judgment of the stop criterion only after enough iterations, so as to avoid the case that the solving procedure is terminated incorrectly due to the slow optimization at the beginning of the problem solving. Since FSMO needs to be called many times, terminating the solving procedure in advance will hardly affect finding the global optimum.

Outer Cutting-Plane Training. After multiple epochs, sometimes there are new constraints generated, but the change of the dual values is extremely small. Then it will bring large computing overhead to continue the iterative procedure, while the accuracy can hardly be improved. Therefore, we add a threshold for the change rate of dual value (e.g., 1‰), so that the training procedure can be terminated in advance.

4 Our FastSSVM Solutions

In this section, we present *FastSSVM*, a parallel and distributed structured SVM training solution that exploits an MPI+OpenMP approach. FastSSVM performs two-level optimizations, including distributed multi-node system level (cf. Section 4.1) and single node level (cf. Section 4.2).

4.1 Distributed Multi-Node System Level Optimizations

We develop a cascade architecture that splits the input data set and solves them separately with multiple MPI processes. The partial results are combined and filtered until the global optimum is reached. The input working set is stored centrally (cf. Section 4.1.1) or locally (cf. Section 4.1.2) to make use of available resources of the distributed system and tackle the challenge of high communication cost between multiple computing nodes. The proof of convergence of the cascade architecture is provided in Section 4.1.3.

4.1.1 FastSSVM with Centralized Storage

As mentioned in Section 2.1, structured SVM training involves two steps: i) working set selection, and ii) solving the QP problem using FSMO. Meanwhile, we add another step of computing the information of each constraint when it is added to the working set, since constraint information is fixed for one specific constraint and is reused many times.

We find that solving the QP problem is often the most expensive part of the training procedure. Therefore, we aim to parallelize the FSMO algorithm. However, parallelization is not trivial due to dependencies between the computation steps. Therefore, we developed a cascade architecture [25], as shown in Figure 3a, where the optimization problem is splitted into smaller and independent sub-problems and the partial solutions are combined in a hierarchical fashion.

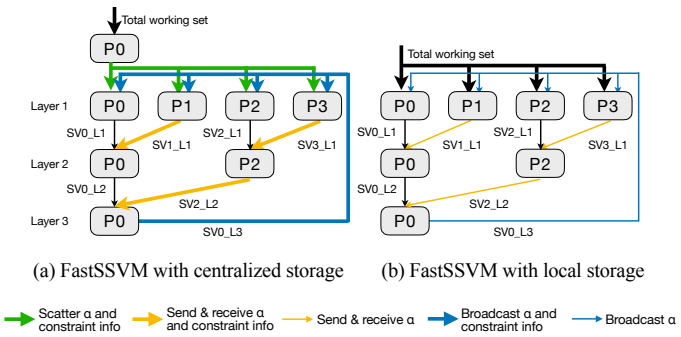


Fig. 3: Overview of FastSSVM. "SVx_Ly" indicates the support vectors produced by process x in layer y.

In the cascade architecture, the root process $P0$ splits the input working set and scatters to all the processes, as shown in green lines in Figure 3a. Each sub-QP problem is then solved by one computing process individually. Then the non-support vectors (i.e., training instances with α value equal to 0) are eliminated to reduce the computation cost, and two sets of partial support vectors are combined for the next layer, as shown in yellow lines. This procedure continues until only one set of support vectors is left. Often such a single pass can produce satisfactory accuracy, however, to reach the global optimum, feedback loops are required. Intuitively, the instances that are recognized as non-support vectors and thus are eliminated by local computations can be added back to the working set in the next feedback loops, if those instances are support vectors globally. Specifically, each process of the first layer receives the resulting support vectors from the last layer, as shown in blue lines, and combined with the non-support vectors of the original input vectors to test for convergence. If the combined support vectors of each process are exactly the same as the input vectors, it has converged to the global optimum. Otherwise, the training procedure needs to proceed with another pass.

The main advantage of the cascade architecture is that it eliminates non-support vectors early from the optimization procedure. In many cases, there are a large number of non-support vectors, thus one single process does not have to compute on the whole working set. Meanwhile, for solving the sub-QP problem on each process, a naive approach is to initialize the weights $\boldsymbol{w}$ to zero vector and iteratively find the solutions. But in the cascade architecture, the weights $\boldsymbol{w}$ of a process can be initialized to better starting points according to the $\boldsymbol{\alpha}$s, when the process receives the combination of support vectors from the previous layer.

For FastSSVM with centralized storge, the working set is stored centrally on the root process, and all the other computing processes do not need to store the entire working set. In the example shown in Figure 3a, *P0* obtains the entire working set as the input of the FSMO algorithm, while the memory storage requirement of processes *P1*, *P2* and *P3* can be significantly reduced in many cases. This can be very suitable for such a scenario where some computing nodes can only provide computing resources but not memory resources. However, such implementation introduces more communication cost. Thus, we propose FastSSVM with local storage to deal with the communication overhead next.

4.1.2 FastSSVM with Local Storage

The efficiency of FastSSVM can be further boosted by optimization on communication. Figure 3a shows the communication between processes, where both point-to-point (i.e., *MPI_Send* and *MPI_Recv*) and collective communication (i.e., *MPI_Scatter* and *MPI_Bcast*) are included.

During the procedure, both the partial solution $\boldsymbol{\alpha}$ and the constraint information need to be transferred. Constraint information includes the margin $\Delta(y_i, \bar{y})$ and the difference of feature vectors $\delta\Psi_i(x_i, \bar{y})$. The main challenge is caused by $\delta\Psi_i(x_i, \bar{y})$. Those vectors are task-dependent and often have a high dimension. For example, the dimension is $N_c \times N_f$ for the multi-class classification problem, where N_c represents the number of classes and N_f represents the number of features. Although they are sparse and we can compress them into the vectors with $2 \times N_f$ dimension, they still cause expensive overhead for communication, considering the large amount of constraints. Communication overhead becomes bottleneck with the increase of parallelism.

To reduce communication overhead, we let all nodes do the working set selection and compute the information of constraints iteratively. For FastSSVM with local storage, the constraint information is stored locally on each process, then the point-to-point and collective communicating of constraint information can be avoided. Hence, only the solution $\boldsymbol{\alpha}$, which is relatively small, needs to be transferred between MPI processes in the cascade architecture, as shown in Figure 3b. Thus the communication cost is significantly reduced while the computation cost almost remains the same.

4.1.3 Convergence Analysis

Here, we theoretically analyze that the cascade architecture is guaranteed to converge to the global optimum.

Let $\mathcal{T}$ be a subset of the working set $\mathcal{S}$ and $L(\mathcal{T})$ denotes the dual value over $\mathcal{T}$ (cf. Problem 2). Let $SV(\mathcal{T})$ be the support vectors from $\mathcal{T}$. We have $\forall \mathcal{T} \subset \mathcal{S}, L(\mathcal{T}) = L(SV(\mathcal{T})) \geq L(\mathcal{S})$. Consider a family $\mathcal{F}$ of sets of training instances, where the set $\mathcal{T}^* \in \mathcal{F}$ is the best set in the family $\mathcal{F}$ that achieves the lowest $L(\mathcal{T})$. We have $L(\mathcal{F}) = L(\mathcal{T}^*) = \min_{\mathcal{T} \in \mathcal{F}} L(\mathcal{T}) \geq L(\mathcal{S})$. In what follows, we first define such a cascade structured SVM architecture that features a sequence of families and then prove its convergence.

Definition 1. A cascade structured SVM architecture is a sequence of families of subsets of working set $\mathcal{S}$, denoted as $\mathcal{F}_m$, which satisfies the following conditions.

1) For all $m > 1$, a set $\mathcal{U} \in \mathcal{F}_m$ contains all the support vectors of the best set $\mathcal{T}^*_{\mathcal{F}_{m-1}} \in \mathcal{F}_{m-1}$.
2) For all m, there is a $k > m$, subject to:
   * All the sets $\mathcal{U} \in \mathcal{F}_k$ contains all the support vectors of the best set in $\mathcal{F}_{k-1}$.
   * The union of all the sets in $\mathcal{F}_k$ is equal to $\mathcal{S}$.

Theorem 1. *A cascade architecture ($\mathcal{F}_m$) converges to the global optimum, i.e., $\exists m^*, \forall m > m^*, L(\mathcal{F}_m) = L(\mathcal{S})$.*

Proof. According to 1) of Definition 1, $SV(\mathcal{T}^*_{\mathcal{F}_{m-1}}) \subset \mathcal{U}$, so we have $L(\mathcal{T}^*_{\mathcal{F}_{m-1}}) = L(SV(\mathcal{T}^*_{\mathcal{F}_{m-1}})) \geq L(\mathcal{U})$, which means $L(\mathcal{F}_{m-1}) = L(\mathcal{T}^*_{\mathcal{F}_{m-1}}) \geq L(\mathcal{U}) \geq L(\mathcal{F}_m)$ for all $m > 1$. Thus, $L(\mathcal{F}_m)$ is monotonically decreasing. This sequence is bounded by $L(\mathcal{S})$, so it converges to some value $L^* \geq W(\mathcal{S})$. There is a $l > 0$ such that for all $m > l$, $L(\mathcal{F}_m) = L^*$. According to 2) of Definition 1, there is a $k > l$, every set $\mathcal{U} \in \mathcal{F}_{k+1}$ contains all the support vectors of the best set $\mathcal{T}^*_{\mathcal{F}_k} \in \mathcal{F}_k$, and $L(\mathcal{F}_{k+1}) = L(\mathcal{F}_k) = L^*$. Following the proof in the literature [25], we know $L(\mathcal{T}^*_{\mathcal{F}_k}) = L(\cup_{\mathcal{U} \in \mathcal{F}_{k+1}})$. Due to that $L(\cup_{\mathcal{U} \in \mathcal{F}_{k+1}} \mathcal{U}) = L(\mathcal{S})$, we can get $L(\mathcal{F}_k) = L(\mathcal{T}^*_{\mathcal{F}_k}) = L(\mathcal{S})$. Since $L(\mathcal{F}_m)$ is monotonically decreasing, for all $m > k$, we have $L(\mathcal{F}_m) = L(\mathcal{S})$. □

Algorithm 3: FastSSVM with separated weights

```
1  Input: S_j, α_{S_j}, C, t, constraint info
2  N ← t
3  repeat
4  #pragma omp parallel num_threads (N)
5    {
6      S_j^{local} ← split S_j into N parts
7      initialize w according to α_{S_j^{local}}
8      repeat
9        for constraint i in S_j^{local} do
10         check the KKT condition
11         if i violates the KKT condition
12           update α_{ji}
13           update w
14         end if
15       end for
16     until no α_{ji} changed during iteration
17   }
18   S_j ← remove non-support vectors
19   N ← N/2
20 until N < 1
```

4.2 Single Node Level Optimizations

In the single node level, multiple OpenMP threads can directly employ the proposed cascade architecture (cf. Section 4.2.1). We also develop techniques to efficiently synchronize the intermediate weights to fully utilize the shared memory systems used by OpenMP (cf. Section 4.2.2).

4.2.1 FastSSVM with separated weights

Inside each MPI process, the OpenMP-based multi-threaded implementation can directly apply to the cascade architecture presented in Section 4.1. The implementation is summarized in pseudo-code in Algorithm 3.

For MPI process j, the partial working set $\mathcal{S}_j$ is the input of all the t OpenMP threads inside the computing node j. Inside the OpenMP region (Line 5 to 17), the partial working set $\mathcal{S}_j$ is splitted into N parts (Line 6), and each thread holds a local weight vector $\boldsymbol{w}$ which is initialize according to the local working set $\mathcal{S}_j^{local}$ (Line 7). Then part of the αs are solved by each thread and the $\boldsymbol{w}$ is updated iteratively until no α_{ji} changed during iteration (Line 8 to 16). After that, non-support vectors are removed from $\mathcal{S}_j$ and then we reduce N to go to the next layer (Line 18 and Line 19). According to Section 4.1.3, the cascade structured SVM architecture can converge to the global optimum.

4.2.2 FastSSVM with shared weights

OpenMP codes typically run on shared memory machines. In a shared memory machine, global memory can be ac-

cessed by all the cores. Information exchanged between threads uses shared variables. Hence, a more efficient implementation is to maintain a shared weight vector in all the threads and to read and write it atomically.

Compared with FastSSVM with separated weights, FastSSVM with shared weights only has one single layer, which saves the execution time. More specifically, the OpenMP parallel region of the separated weights version (Line 5 to 17) is executed $log_2 t$ times as shown in Algorithm 3, where t represents the number of OpenMP threads. For FastSSVM with shared weights, the parallel region is only executed once with a parallel factor t as the outer loop (Line 3, Line 19 to 20) is eliminated. Since the weight vector is shared by all the threads for FastSSVM with shared weights, we initialize the global weight vector outside the parallel region, instead of initializing the local weight vector inside the parallel region (Line 7) for FastSSVM with separated weights. Meanwhile, we define a critical area for the update of the weight vector (Line 13) using the compiler directive #*pragma omp critical* to ensure that only one thread can read or write the shared weights at the same time.

4.3 Performance Analysis

The FSMO implementation is the core of FastSSVM, and the cascade architecture used in our FSMO implementation causes parallelization overhead that grows with the number of processes p. For example, these cascade architectures cannot run faster than $\Omega(logp)$. Therefore, we mainly analyze our implementation of FSMO by estimating overhead of the cascade architecture in this section, and suggest the usage of multiple nodes to solve the QP problem.

The FSMO algorithm needs to be invoked many times during the training. Assume that M is the average number of constraints to be handled in each call. FSMO iteratively traverses the constraints and updates the corresponding αs of the constraints, until no α is updated after I iterations.

In the case of single-node implementation, the operation on the critical path is executed $M \times D$ times in each iteration, where D is the dimension of the difference between feature vectors $\delta\Psi_i(x_i, \bar{y})$. Assuming that the execution of such an operation needs t seconds, then the total time of I iterations is $T_{total}(1) = I \times t \times M \times D$.

For a L-layer p-node cascade architecture with $L = log_2 p + 1$, suppose that the ratio of support vectors (SVs) is k, which means there are $k \times M$ SVs in total in the training problem. Those SVs are filtered through each layer. Assume that the first $L-1$ layers have the same filtering ratio while the last layer handles all the SVs. Specifically, each node in each layer passes SVs with a same ratio of ρ to the next layer, where $\rho = k^{\frac{1}{L-1}}$. Then, in the i-th layer, $\frac{p}{2^{i-1}}$ nodes handle $\rho^{i-1} \times M$ constraints in parallel, so the computation time of each node in layer i is:

$$T_{each}^{comp}(i) = I \times t \times \frac{\rho^{i-1} \times M}{\frac{p}{2^{i-1}}} \times D \tag{4}$$

For each node of the first $L-1$ layers, the indices and values of αs of $\frac{\rho^i \times M}{\frac{p}{2^{i-1}}}$ SVs are passed to the next layer. We denote the network transmission speed by E bps. Then, the communication time of each node of layer i ($i \neq L$) is:

$$T_{each}^{comm}(i) = I \times \frac{\rho^i \times M}{\frac{p}{2^{i-1}}} \times \frac{96}{E} \tag{5}$$

where 96 is the number of bits needed to transmit an SV (i.e., 4 bytes for the index of α and 8 bytes for the value of it). From the above results, we can draw three conclusions.

First, the execution time of the FSMO algorithm increases as the number of constraints M and the number of features N_f increase. According to Equation (4), $T_{each}^{comp}(i)$ is proportional to M and D, where the value of D depends on the task (e.g., $D = 2 \times N_f$ in our implementation).

Second, according to Equations (4) and (5), we get the total time of I iterations of the L-layer cascade architecture:

$$T_{total}(L) = \begin{cases} \frac{I \cdot M}{2^{L-1}}(t \cdot D \cdot \frac{1-(2\rho)^L}{1-2\rho} + \rho \cdot \frac{96}{E} \cdot \frac{1-(2\rho)^{L-1}}{1-2\rho}), & \rho \neq 0.5 \\ \frac{I \cdot M}{2^{L-1}}(t \cdot D \cdot L + \frac{1}{2} \cdot \frac{96}{E} \cdot (L-1)), & \rho = 0.5 \end{cases}$$

where $\rho = k^{\frac{1}{L-1}}$. If $T_{total}(L+1) - T_{total}(L) < 0$, it means that the execution time is shortened when we increase the number of nodes p from 2^{L-1} to 2^L.

Third, it is not recommended to increase nodes if k is large. According to Equation (4), we can get:

$$T_{each}^{comp}(i+1) - T_{each}^{comp}(i) = I \times t \times \frac{M}{p} \times D \times ((2 \times \rho)^i - (2 \times \rho)^{i-1})$$

When $\rho > 0.5$ (i.e., $k > 0.5^{L-1}$), $T_{each}^{comp}(i+1) - T_{each}^{comp}(i) > 0$. In other words, when the number of SVs is large, the computation of last layer becomes the bottleneck. More precisely, if the SV ratio $k > 50\%$ for a 2-layer cascade, or $k > 25\%$ for a 3-layer cascade, or $k > 12.5\%$ for a 4-layer cascade and so on, the computation time of the last layer is the longest, which is $T_{each}^{comp}(L) = I \times t \times k \times M \times D = k \times T_{total}(1)$. Therefore, with the increase of k, $T_{each}^{comp}(L)$ approaches the total time of the 1-layer cascade $T_{total}(1)$, and extra communication overhead is also added.

This theoretical analysis provides a general guideline on multiple nodes. The situation in the experiments is often more complicated. The values of M, I, D, and k all depends on the specific problem to be solved, and they are usually unknown beforehand. In practice, the number of iterations (i.e., I) for FSMO solver on each node in each layer is actually different. The fastest node must wait for the slowest node to finish its iterations. Besides, it is only an ideal case that each node in each layer holds the same filtering ratio ρ. After all, the number of the resulting SVs for different nodes in the same layer is different. It results in different communication times and directly leads to different problem scales for different nodes in the subsequent layers. Moreover, the number of SVs in different layers is often different.

5 Experimental Evaluation

In this section, we first present our experimental setup (cf. Section 5.1), then present the main result by comparing FastSSVM with the state-of-the-art work (cf. Section 5.2). After that, we compare different OpenMP and MPI implementations of FastSSVM and determine the best ones (cf. Section 5.3). Finally, we compare the performance of FastSSVM under different configurations (cf. Section 5.4).

5.1 Experimental Setup

We implemented FastSSVM using hybrid MPI and OpenMP parallellism in C++ for three kinds of applications, including multi-class classification, multi-label classification and object detection. For comparison, we ran SVM-Struct [6]

TABLE 1: Data set information and parameters. "MC" indicates Multi-Class task, "ML" indicates Multi-Label task, and "OD" indicates Object Detection task. The size of bounding box for object detection is showed in the "# feature" column.

Data set	Task	# instance	# class	# feature	$C(\times 1000)$
letter	MC	15,000	26	16	100
MNIST	MC	60,000	10	780	10
pendigits	MC	119,904	10	16	10
protein	MC	17,766	3	357	10
acoustic	MC	78,823	3	50	100
seismic	MC	78,823	3	50	100
smallNORB	MC	24,300	5	2,048	10
tmc2007	ML	43,038	22	30,438	1,000
digit	OD	20,000	10	784	100

that uses the Hildreth and D'Espo optimizer [26] for solving QP problems and we also implemented the original FSMO algorithm [19]. Specially, for object detection tasks, we used the API of SVM-Struct and specialized some functions to implement our own instantiation. We ran our experiments on a cluster of eight computing nodes connected by 1Gpbs Ethernet, each with a quad-core 2.67GHz Intel i7920 CPU and 8GB RAM. The value $\varepsilon = 0.1$ was used for all the experiments as the stopping criterion (cf. Algorithm 1 Line 7). We used publicly available data sets *letter*, *MNIST*, *pendigits*, *protein*, *SensIT Vehicle (acoustic)*, *SensIT Vehicle (seismic)*, *smallNORB* and *siam-competition2007 (tmc2007)*. Besides, for object detection task, we created a *handwritten digit detection data set (digit)* from the MNIST of handwritten digits [27] by sampling 20000 images from the original data set and putting each of the sampled images into a random location of a 64 * 64 image. The accuracy and training time increase as the increase of the regularization constant C. We chose C from 1000, 10000, 100000 and 1000000 for each data set, so that the selected appropriate C leads to satisfactory accuracy with reasonable training time. Table 1 summarizes the information of the data sets and the parameter C we set.

5.2 Overall Effectiveness

In this section, we mainly provide the overall effectiveness of our FastSSVM. We first compare FastSSVM with the existing work (cf. Section 5.2.1), then investigate the impact of different optimizations (cf. Section 5.2.2), and finally examine the single node efficiency (cf. Section 5.2.3).

5.2.1 Comparison with State-of-the-art Work

We conducted experiments for FastSSVM, SVM-Struct [6] and FSMO [19], and Table 2 shows the performance comparison in terms of training time and accuracy. Detailed process of determining the best optimization configuration for FastSSVM can be found in Sections 5.3 and 5.4. We used the default OpenMP and MPI implementations, i.e. FastSSVM with shared weights and local storage, due to their good efficiency as demonstrated in Section 5.3. We also set the number of feedback iterations of the cascade architecture to 1 as demonstrated in Section 5.4.2. Different data sets get the shortest training time under different configurations as shown in Section 5.4.1. For example, *letter* gets the shortest training time when p is 4 and t is 8, while *seismic* benefits from $p = 8$ and $t = 8$. We set the best configuration of

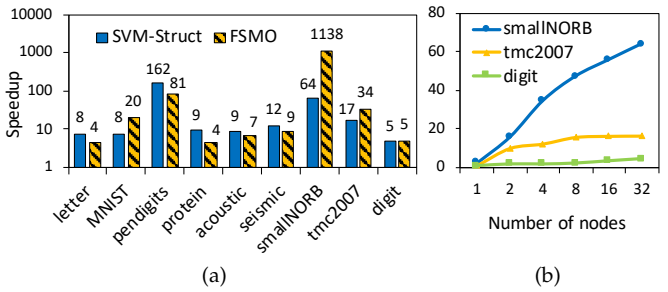


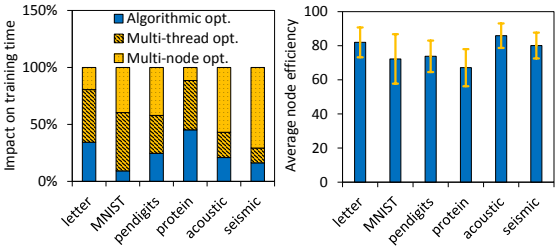
Fig. 4: Comparsion of training time: (a) shows the speedups of FastSSVM over existing implementations; (b) shows the speedups of FastSSVM under different number of computing nodes over SVM-Struct.

p and t for each data set and the results are shown in the "FastSSVM-parallel on multiple nodes" column.

Comparison of training time. For ease of comparison, Figure 4a shows the speedups of FastSSVM over SVM-Struct and FSMO on all the data sets tested. Generally, FastSSVM achieves four to 34 times speedup compared with the existing work, and has up to 162 times speedup compared with SVM-Struct and even more than 1000 speedup over FSMO on the *smallNORB* data set. The speedups are mainly due to the two-level optimizations for parallel and distributed training and the techniques of adapted batch size and training convergence improvement, which are not used in [6], [19]. More specifically, Figure 4b shows the general speedups of FastSSVM under different number of computing nodes over the existing work SVM-Struct. Multiple computing nodes can speedup up the training procedure, indicating that FastSSVM has good scalability. Detailed experiments on the effect of optimization configurations can be found in Section 5.4.

Comparison of accuracy. We used accuracy as evaluation indicators for multi-class problems, F1 score for multi-label problems. For object detection, we used IoU (i.e., Intersection over Union)-based indicators. An example is viewed as a correct example if it has the correct predicted label and the IoU between the ground-truth and the predicted bounding box is larger than 0.5. We can observe that all the implementations have the similar accuracy, indicating that FastSSVM can achieve higher efficiency while maintaining the similar accuracy compared with the existing work.

MNIST and *pendigits* are well-known real-world data sets and ideal testbeds for new machine learning methods, and thus have been extensively used to explore the performance of neural networks and other deep learning techniques. DropConnect [28] and MIN [29] are state-of-the-art for *MNIST* with and without data augmentation or preprocessing, and obtain the accuracies of 99.79% and 99.76%, respectively; ShapeNet [30] and KerNet [31] are the most recent work tested on *pendigits* and obtain the accuracies of 97.7% and 96.71%, respectively. We observe that there are some gaps in terms of accuracy between FastSSVM and state-of-the-art neural network based solutions. However, although it has become non-challenge for neural network based solutions to achieve high accuracies, such networks are often big and deep with large model



(a) Individual optimizations (b) Average node efficiency

Fig. 5: Impact of the individual optimizations and average node efficiency.

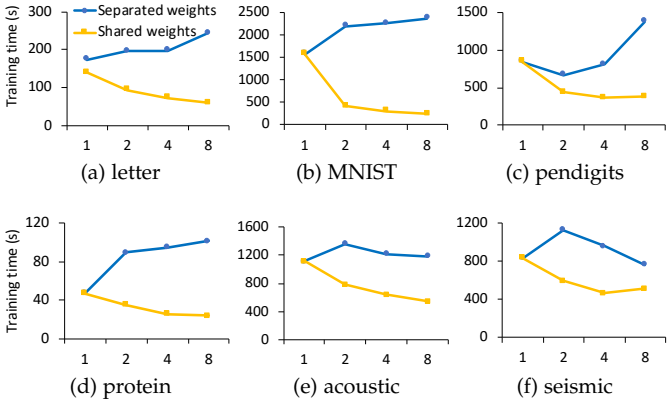


Fig. 6: Comparison of OpenMP-based implementations under different number of OpenMP threads.

sizes and a huge number of hyper-parameters, which results in weeks or months to train them on CPUs, and tuning hyper-parameters requires even more extra time. Therefore, such models are often required to be trained on modern massively parallel GPUs. We have conducted experiments on *MNIST* for some small networks, such as LeNet [32], on the same CPU used in our other experiments. Results show that even some small networks require more training time than FastSSVM. Moreover, FastSSVM enjoys a much smaller model size and much fewer hyper-parameters. For example, the model size of *MNIST* is about 60KB, while it requires 200KB even for the relatively small neural network LeNet.

5.2.2 Impact of Individual Optimizations

As we have discussed in Sections 3 and 4, we have some optimizations for our FastSSVM. Here, we study their impacts on the overall efficiency. The optimizations include (i) Algorithmic optimizations: include adapted batch size (cf. Section 3.1) and training convergence improvement (cf. Section 3.2) to deal with a large number of constraints and inferior convergence rates; (ii) Multi-thread optimization: uses OpenMP to utilize the shared memory system (cf. Section 4.2); (iii) Multi-node optimization: uses MPI to efficiently communicate given the available resources (cf. Section 4.1). To investigate where the speedups of FastSSVM to FSMO originate from, we first switched off the multi-node optimization under the "FastSSVM-parallel on multiple nodes", and the results under the best configuration of t are shown in the "FastSSVM-parallel on a single node" column. Then, we switched off the multi-thread optimization, and the results are shown in the "FastSSVM-sequential" column.

Figure 5a shows the contribution of each optimization to the overall speedups. Overall, 25% of the speedup originates from algorithmic optimizations, 35% from multi-thread optimization, and 40% from multi-node optimization.

5.2.3 Single Node Efficiency

For each data set, we further examine the training time of each single node (i.e., MPI process) under its best configuration of p and t to illustrate the node efficiency. For ease of comparison and result visualization, we normalize the total training time to 100 for each data set and compute the average and standard deviation of the training time on all the single nodes. The results are shown in Figure 5b. We use the error bar to show the standard deviation.

In general, the average node efficiency is higher than 70% and the deviation is basically less than 10% on the data sets tested. In practice, the average node efficiency cannot achieve 100%, due to a certain deviation of workload between different nodes. More specifically, different nodes have different workloads in the step of solving QP problem using FSMO, due to the inherent characteristics of the cascade structured SVM architecture. Taking a 4-node 3-layer cascade architecture as an example (cf. Figure 3), $P0$ has to do the computation in all the 3 layers and hence has the heaviest workload; $P2$ computes in 2 layers and has the moderate workload; while $P1$ and $P3$ only need to compute in the first layer and hence have the lightest workloads. However, the time of the QP solving step often can be significantly reduced under the best configuration, and hence such limitation of the cascade architecture in the QP solving step does not have much impact.

5.3 Determining the Best OpenMP/MPI Implementation

In this section, we aim at determining the best OpenMP implementation (cf. Section 5.3.1) and the best MPI implementation (cf. Section 5.3.2), which serve as the default configuration in the following experiments.

5.3.1 OpenMP: Separated Weights VS Shared Weights

Here, we compare the performance of our two OpenMP-based implementations. The experiments were conducted on a single node, and the number of OpenMP threads t ranges from 1 to 8. Figure 6 illustrates the training time for different t for both OpenMP implementations, i.e., FastSSVM with separated weights (cf. Section 4.2.1) and FastSSVM with shared weights (cf. Section 4.2.2). We observe that the latter always leads to shorter training time, indicating that we can employ the weight sharing approach to take more efficient advantage of the shared memory systems used by OpenMP. On some data sets such as *MNIST* and *protein*, the improvement is more than 4 times. In the following experiments, the default OpenMP implementation is shared weights due to its good efficiency.

5.3.2 MPI: Centralized Storage VS Local Storage

We compare the performance of our two MPI-based implementations here. We set the number of OpenMP threads t to 8 to fully utilize the computing power of each node, and

TABLE 2: Training elapsed time and accuracy of FastSSVM and comparison with existing solutions.

Data set	FastSSVM-parallel on multiple nodes		FastSSVM-parallel on a single node		FastSSVM-sequential		SVM-Struct		FSMO	
	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy
letter	48.1s	72.20%	59.7s	72.20%	140s	72.12%	372s	69.90%	213s	71.68%
MNIST	144s	92.07%	239s	92.00%	1574s	91.96%	1121s	91.91%	2946s	91.83%
pendigits	208s	90.54%	360s	90.68%	846s	90.60%	33841s	90.88%	16789s	90.79%
protein	21.5s	68.54%	24.3s	68.81%	47.5s	68.81%	201s	68.81%	96s	68.82%
acoustic	233s	67.07%	540s	66.46%	1112s	66.01%	2011s	65.86%	1664s	64.15%
seismic	135s	66.5%	463s	67.03%	836s	66.93%	1681s	66.65%	1245s	64.32%
smallNORB	278s	75.21%	1636s	75.57%	6045s	75.39%	17777s	75.80%	316602s	75.97%
tmc2007	215s	70.65%	340s	70.36%	3144s	68.17%	3565s	63.64%	7256s	68.17%
digit	105523s	64.52%	229877s	66.53%	478596s	67.45%	499189s	66.56%	527474s	69.89%

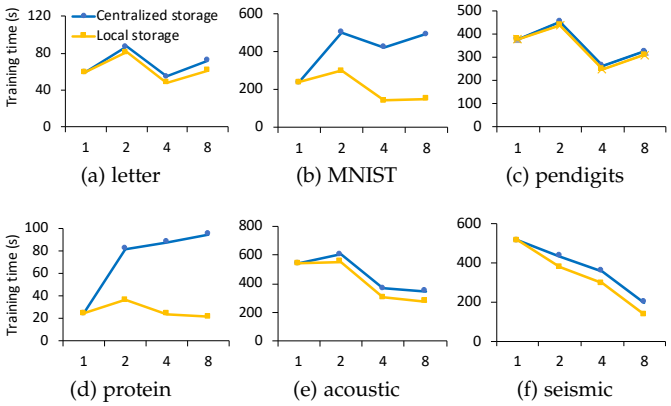


Fig. 7: Comparison of MPI-based implementations under different number of MPI processes.

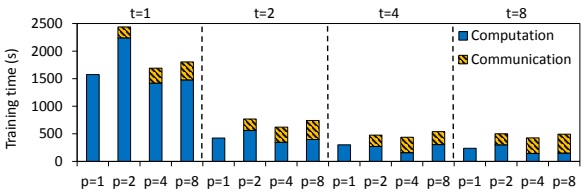


Fig. 8: Computation and communication time of FastSSVM with centralized storage and shared weights on MNIST.

the number of MPI processes p ranges from 1 to 8. Figure 7 shows the training time for the two MPI implementations on six of the data sets. We observe that FastSSVM with local storage always leads to higher efficiency on the data sets tested, because this implementation leverages more memory to save communication cost. We have the same observation on all the data sets tested. Particularly, the training time of the two implementations is almost the same for the *pendigits* data set, as shown in Figure 7c, because the dimension N_f of *pendigits* is rather small compared with the other data sets. As discussed in Section 4.1.2, communication cost is mainly caused by transferring the vector $\delta\Psi_i(x_i, \bar{y})$, whose dimension is $2 \times N_f$ in our implementation. Thus, communication cost of *pendigits* in FastSSVM with centralized storage is relatively small, hence the improvement in FastSSVM with local storage has a limited effect on the training time. On the contrary, we can observe a notable difference between the two MPI implementations for the *MNIST* and *protein* data sets (cf. Figures 7b and 7d), whose N_f is relatively large.

Communication study. We further investigate the effect of communication cost on FastSSVM with centralized storage. We varied the number of MPI processes p and the number of OpenMP threads t. Figure 8 shows the computation and communication time for varying parameter settings with the *MNIST* data set as an example. We observed that the communication overhead increases with the increase of p. When $p = 8$ and $t = 8$, the total training time is 493.9s, while the communication time occupies 342.5s. Communication time accounts for more than half of the total time, becoming the bottleneck of the whole training procedure. Most of the communication cost is due to transferring $\delta\Psi_i(x_i, \bar{y})$ between adjacent layers, as shown in Figure 3a. The other data sets also have similar trends. When p reaches 8, there is usually a non-negligible communication overhead. Thus, some of the data sets tested achieve their shortest training time when $p = 1$ for the centralized storage version. Therefore, we develop FastSSVM with local storage to eliminate the communication of $\delta\Psi_i(x_i, \bar{y})$, which reduces the communication cost by over 90% and shortens the whole training time. For example, in the case of $p = 8$ and $t = 8$ for *MNIST*, the communication time of the FastSSVM with local storage is less than 1s. Therefore, compared with FastSSVM with centralized storage, FastSSVM with local storage reduces more than 50% of the training time of *MNIST*.

5.4 Effect of Optimization Configurations

In this section, we first extensively compare the training time under different parallel granularity and find the shortest training time that FastSSVM can achieve (cf. Section 5.4.1). Then, we examine the effect of feedback iterations of the cascade architecture on the convergence behavior and training time (cf. Section 5.4.2).

5.4.1 Effect of Parallel Granularity

For each data set, we conducted experiments for FastSSVM with varying number of MPI processes p and OpenMP threads t. We used the FastSSVM with local storage and shared weights, which achieves the best efficiency. Considering the problem scales, we extend p from eight to 32 for experiments on the *smallNORB*, *tmc2007* and *digit* data sets[1], while eight computing nodes are enough for other data sets. Figure 9 illustrates the training time for varying p and t on

1. Since the original cluster only has eight computing nodes, we ran experiments on these three data sets on Oracle Cloud Infrastructure. We created 32 computing instances, each with a 8-core 2.00GHz Intel Xeon Platinum 8167M CPU.

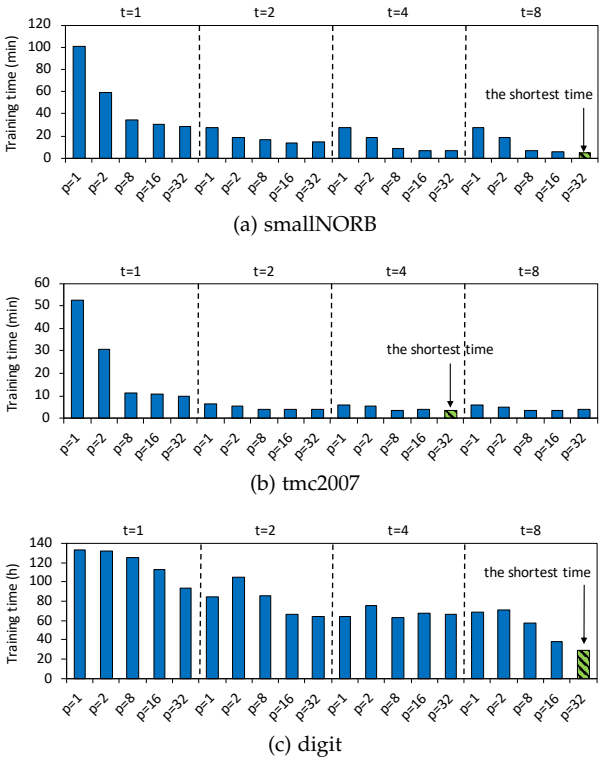


Fig. 9: Training time on different parallel granularity for FastSSVM with local storage and shared weights.

these three data sets. The other data sets also have similar trends. We have the two key findings described below.

Varying number of OpenMP threads. Multiple OpenMP threads can reduce the training time in most of the cases, indicating our techniques to take advantage of the computing power of multi-cores. In particular, when t increases from 1 to 2, the training time is significantly reduced; when t continues to increase to 4 or 8, the training time tends to decrease slowly and gradually stabilize. In our experiments, six out of nine data sets achieve the shortest training time when t reaches 8. The exceptions include the *tmc2007*, *pendigits* and *acoustic* data sets which achieve the shortest training time when $t = 4$. In the case of the three data sets, the training time is similar when $t = 4$ and $t = 8$.

Varying number of MPI processes. An appropriate number of MPI processes can reduce the training time, indicating that FastSSVM can be efficiently scaled to multiple computing nodes. Figure 9 shows that the three data sets achieve the shortest training time when $p = 32$; other data sets also achieve the shortest training time when p reaches 4 or 8. In particular, when the number of MPI processes p increases from 1 to 2, the training time increases in some cases. One reason is that in the two-layer cascade structured SVM architecture, the procedure of non-support vector elimination is only performed twice; when p reaches 4 and 8, this procedure is performed 6 times and 14 times, respectively, thus we can usually obtain a shorter training time. With the increase of p, the training time tends to gradually stabilize, because the main limitation of the cascade architecture is the last layer that consists of one single sub-QP optimization procedure and its size has a lower bound given by the number of support vectors.

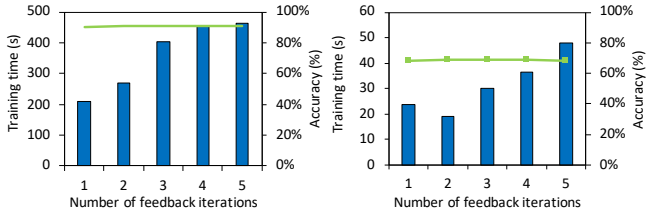


Fig. 10: Accuracy and training time over # of feedback iterations of FastSSVM with local storage and shared weights.

5.4.2 Effect of Feedback Iterations

We examine the effect of feedback iterations of the cascade structured SVM architecture on the convergence behavior and execution time. We set the best configuration of p and t for each data set and the maximum number of feedback iterations N_m varies from one to five. We use the accuracy to show the final loss of different N_m. Figure 10 uses two example data sets to show the accuracy and training time trend for different number of iterations of the cascade architecture. The implementation we used is FastSSVM with local storage and shared weights. We observe that with the increase of N_m, the training time increases in most of the cases, while the accuracy maintains similar. The exceptions are *protein* and *seismic* data sets, where the training time does not increase monotonically with N_m. The main reason is that as N_m increases, the better convergence leads to a decrease in the number of epochs (where an epoch represents one complete pass through the training data set). Taking *protein* (cf. Figures 10b) as an example, when the number of feedback iteration $N_m = 1$, the training procedure spends 8 epochs; when N_m reaches 2, it only takes 7 epochs for the training procedure, reducing the total training time to some extent. In general, practically a single pass through the cascade architecture can produce sufficient accuracy. Therefore, we serve one iteration without feedback loops as the default configuration. As one iteration in the cascade architecture is computationally expensive, our solution tends to be optimal because the minimal number of iteration is needed. This provides a relatively simple way for solving problems.

6 Related Work

Machine learning has been successful in many applications in recent years [33], [34], and high-performance computing has played a key role in this success. This study mainly focuses on improving the efficiency of a machine learning algorithm: structured SVMs. Structured SVM generalizes the traditional SVM to new formulations that has structured outputs, thus can be used to solve structured prediction tasks in a number of areas. In what follows, we categorize the most relevant related work into three categories: the studies dedicated to training SVMs, the studies dedicated to training structured SVMs, and the studies dedicated to solving structured prediction problems in general.

Training SVMs. Platt [35] proposed the SMO algorithm which is simple and efficient, and hence SMO is used in LibSVM [36], WEKA and ThunderSVM [17]. Other studies [37], [38], [39] in parallel or distributed SVM training

have been proposed for improving the efficiency of SVMs. Catanzaro et al. [40] first introduced GPUs for SVM training. Wen et al. [41], [42] proposed GPU based SVMs with precomputing the whole kernel matrix which is stored in high-speed storage (e.g., SSDs). Another study [43] further improves the algorithms by avoiding storage of the whole kernel matrix. However, these studies are to optimize ordinary SVMs which aim to solve binary classification and regression problems. Those approaches are not applicable to complex structured output problems.

Training structured SVMs. Some approaches [44], [45], [46] have been proposed for efficient structured SVM learning. For example, Balamurugan et al. [46] proposed SDM, which traverses the training data set sequentially and optimizes the dual variable corresponding to one instance at a time. Chang et al. [47] proposed a parallel learning algorithm DEMI-DCD. It decouples the model update steps from the inference steps during learning. However, it solves the inference problems in parallel while using one thread for learning. This approach does not deal with the large number of constraints and may be slow for large data sets. Since 2006, the community has been overwhelmed by deep learning, particularly DNN based techniques. Little research effort has been dedicated to traditional machine learning. Structured SVMs provide approaches to solve complex problems in many domains. This work focuses on improve the efficiency of structured SVM training to boost the applications of structured SVMs.

Solving structured prediction problems. Various approaches have been proposed for solving structured prediction tasks. Like structured SVMs, some approaches are the extension of the standard classification methods [48]. And some methods are the integration with deep learning [49]. There are also some search based methods for learning structured output models. Those methods learn different forms of knowledge, including greedy policies [50], heuristic and cost functions [51], coarse-to-fine knowledge [52], and so on. This work aims to specifically improve the training efficiency of structured SVMs.

7 Conclusion and Future Work

In this paper, we have developed an efficient solution for structured SVM training and elaborated the significance of accelerating the training procedure of structured SVMs. The main challenges of developing a fast solution for training structured SVMs include (i) a large number of constraints for the QP problem, (ii) inferior convergence rates for large and complex structured problems, and (iii) high communication cost to transfer intermediate results on a distributed multi-core system. FastSSVM leverages a series of techniques to alleviate the challenges and achieve high efficiency, including adapted batch size, training convergence improvement, local storage of constraint information, and sharing intermediate weights. Our experiments on publicly available data sets have shown that FastSSVM outperforms the existing solutions by at least four times, and by two to three orders of magnitude in some cases, indicating the effectiveness of our solution on distributed computer systems. This work can potentially boost the popularity of structured SVMs—a fundamental machine learning algorithm.

The total training time of FastSSVM can be further reduced by carefully optimizing the batch size in working set selection, and we consider this direction as our future work. Moreover, FastSSVM spends much time on traversing all possible constraints. We will further investigate effective approximation methods for improving FastSSVM on large and complex problems.

Acknowledgement

This work is supported by the National Key R&D Program of China (Grant No. 2020AAA0103800), a MoE AcRF Tier 1 grant (T1 251RES1824) in Singapore, the National Natural Science Foundation of China (Grant No. 62072186), and the Guangdong Basic and Applied Basic Research Foundation (Grant No. 2019B1515130001). This research is also supported by Oracle for Research, Australia.

References

[1] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *PAMI*, 2021.
[2] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *PAMI*, 2020.
[3] S. Sivapatham, R. Ramadoss, A. Kar, and B. Majhi, "Monaural speech separation using ga-dnn integration scheme," *Applied Acoustics*, vol. 160, p. 107140, 2020.
[4] S. Sivapatham, A. Kar, and R. Ramadoss, "Performance analysis of various training targets for improving speech quality and intelligibility," *Applied Acoustics*, vol. 175, p. 107817, 2021.
[5] M. E. Mavroforakis and S. Theodoridis, "A geometric approach to support vector machine (svm) classification," *IEEE transactions on neural networks*, vol. 17, no. 3, pp. 671–682, 2006.
[6] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *ICML*, p. 104, 2004.
[7] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *JMLR*, vol. 6, no. Sep, pp. 1453–1484, 2005.
[8] M. Le Nguyen, A. Shimazu, and X.-H. Phan, "Semantic parsing with structured svm ensemble classification models," in *COLING/ACL 2006 Main Conference Poster Sessions*, pp. 619–626, 2006.
[9] M. K. Sharma and V. S. Dhaka, "Segmentation of handwritten words using structured support vector machine," *PAA*, pp. 1–13, 2019.
[10] S. Nowozin and C. H. Lampert, "Structured learning and prediction in computer vision," *Foundations and Trends® in Computer Graphics and Vision*, vol. 6, no. 3–4, pp. 185–365, 2011.
[11] S. Branson, O. Beijbom, and S. Belongie, "Efficient large-scale structured learning," in *CVPR*, pp. 1806–1813, 2013.
[12] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. L. Hicks, and P. H. Torr, "Struck: Structured output tracking with kernels," *PAMI*, vol. 38, no. 10, pp. 2096–2109, 2015.
[13] J. Ning, J. Yang, S. Jiang, L. Zhang, and M.-H. Yang, "Object tracking via dual linear structured svm and explicit feature map," in *CVPR*, pp. 4266–4274, 2016.
[14] D. Li, Y. Ju, and Q. Zou, "Protein folds prediction with hierarchical structured svm," *Current Proteomics*, vol. 13, no. 2, pp. 79–85, 2016.
[15] W. Lin, D. Ji, and Y. Lu, "Disorder recognition in clinical texts using multi-label structured svm," *BMC bioinformatics*, vol. 18, no. 1, p. 75, 2017.
[16] Y. Zhu, X. Zhu, M. Kim, D. Shen, and G. Wu, "Early diagnosis of alzheimer's disease by joint feature selection and classification on temporally structured support vector machine," in *MICCAI*, pp. 264–272, Springer, 2016.
[17] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen, "Thundersvm: A fast svm library on gpus and cpus," *JMLR*, vol. 19, no. 1, pp. 797–801, 2018.
[18] C.-p. Lee, K.-W. Chang, S. Upadhyay, and D. Roth, "Distributed training of structured svm," *arXiv preprint arXiv:1506.02620*, 2015.
[19] C. Lee and M.-G. Jang, "Fast training of structured svm using fixed-threshold sequential minimal optimization," *ETRI journal*, vol. 31, no. 2, pp. 121–128, 2009.

[20] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *IJCV*, vol. 88, no. 2, pp. 303–338, 2010.

[21] E. Charniak, B. D, N. Ge, K. Hall, and M. Johnson, "Bllip 1987-89 wsj corpus release 1 ldc2000t43," *Philadelphia: Linguistic Data Consortium*, 2000.

[22] T. Joachims, T. Finley, and C.-N. J. Yu, "Cutting-plane training of structural svms," *Machine learning*, vol. 77, no. 1, pp. 27–59, 2009.

[23] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[24] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[25] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, "Parallel support vector machines: The cascade svm," in *NIPS*, pp. 521–528, 2005.

[26] I. N. Bronshtein and K. A. Semendyayev, *Handbook of mathematics*. Springer Science & Business Media, 2013.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *ICML*, pp. 1058–1066, PMLR, 2013.

[29] J.-R. Chang and Y.-S. Chen, "Batch-normalized maxout network in network," *arXiv preprint arXiv:1511.02583*, 2015.

[30] G. Li, B. Choi, J. Xu, S. S. Bhowmick, K.-P. Chun, and G. L. Wong, "Shapenet: A shapelet-neural network approach for multivariate time series classification," in *AAAI*, 2021.

[31] I. Lauriola, C. Gallicchio, and F. Aiolli, "Enhancing deep neural networks via multiple kernel learning," *Pattern Recognition*, vol. 101, p. 107194, 2020.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] K. Zhang, A. Guliani, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman, "Machine learning-based temperature prediction for runtime thermal management across system components," *TPDS*, 2017.

[34] M. Grossman, M. Breternitz, and V. Sarkar, "Hadoopcl2: Motivating the design of a distributed, heterogeneous programming system with machine-learning applications," *TPDS*, vol. 27, no. 3, pp. 762–775, 2016.

[35] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.

[36] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM TIST*, vol. 2, pp. 27:1–27:27, 2011.

[37] J.-P. Zhang, Z.-W. Li, and J. Yang, "A parallel svm training algorithm on large-scale classification problems," in *International Conf. on Machine Learning and Cybernetics*, vol. 3, pp. 1637–1641, 2005.

[38] F. Ö. Çatak and M. E. Balaban, "A mapreduce-based distributed svm algorithm for binary classification," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 24, no. 3, pp. 863–873, 2016.

[39] A. Navia-Vázquez, D. Gutierrez-Gonzalez, E. Parrado-Hernández, and J. Navarro-Abellan, "Distributed support vector machines," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, p. 1091, 2006.

[40] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast Support Vector Machine training and classification on graphics processors," in *ICML*, pp. 104–111, 2008.

[41] Z. Wen, R. Zhang, K. Ramamohanarao, J. Qi, and K. Taylor, "MASCOT: fast and highly scalable SVM cross-validation using GPUs and SSDs," in *ICDM*, pp. 580–589, 2014.

[42] Z. Wen, R. Zhang, K. Ramamohanarao, and L. Yang, "Scalable and fast SVM regression using modern hardware," *WWW*, vol. 21, no. 2, pp. 261–287, 2018.

[43] Z. Wen, J. Shi, B. He, J. Chen, and Y. Chen, "Efficient multi-class probabilistic svms on gpus," *TKDE*, vol. 31, no. 9, pp. 1693–1706, 2018.

[44] T. Finley and T. Joachims, "Training structural svms when exact inference is intractable," in *ICML*, pp. 304–311, 2008.

[45] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher, "Block-coordinate frank-wolfe optimization for structural svms," in *ICML*, pp. 53–61, 2013.

[46] P. Balamurugan, S. Shevade, S. Sundararajan, and S. S. Keerthi, "A sequential dual method for structural SVMs," in *SDM*, pp. 223–234, 2011.

[47] K.-W. Chang, V. Srikumar, and D. Roth, "Multi-core structural svm training," in *ECML/PKDD*, pp. 401–416, Springer, 2013.

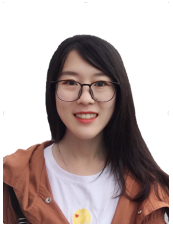
[48] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[49] D. Belanger, B. Yang, and A. McCallum, "End-to-end learning for structured prediction energy networks," *arXiv preprint arXiv:1703.05667*, 2017.

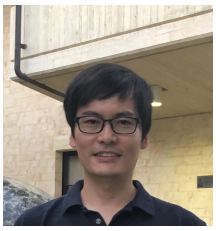
[50] K.-W. Chang, A. Krishnamurthy, A. Agarwal, H. Daume, and J. Langford, "Learning to search better than your teacher," in *ICML*, pp. 2058–2066, 2015.

[51] J. R. Doppa, A. Fern, and P. Tadepalli, "Hc-search: A learning framework for search-based structured prediction," *Journal of Artificial Intelligence Research*, vol. 50, pp. 369–407, 2014.

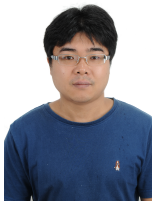
[52] D. Weiss and B. Taskar, "Structured prediction cascades," in *International Conference on Artificial Intelligence and Statistics*, pp. 916–923, 2010.



Jiantong Jiang is currently a PhD student at The University of Western Australia, after receiving a Master's degree from Northeastern University in China. Before commencing her PhD study, she was a Research Assistant at School of Software Engineering, Zhejiang University, China. Her research interests include high-performance computing and machine learning.



Zeyi Wen is a Lecturer at The University of Western Australia (UWA). Before joining UWA, Zeyi worked as a Research Fellow in National University of Singapore from 2017 and 2019, after receiving his PhD degree from and working as a Research Fellow at The University of Melbourne. Zeyi's areas of research include parallel computing, machine learning and data mining.



Zeke Wang received his Ph.D. degree from Zhejiang University, China in 2011. He is a Research Professor at Collaborative Innovation Center of Artificial Intelligence, Department of Computer Science, Zhejiang University, China. His current research interests mainly focus on building machine learning systems using FPGAs.

Bingsheng He received the bachelor degree in computer science from Shanghai Jiao Tong University (1999-2003), and the PhD degree in computer science in Hong Kong University of Science and Technology (2003-2008). He is an Associate Professor in School of Computing of National University of Singapore. His research interests are high performance computing, distributed and parallel systems, and database systems.

Jian Chen is currently a Professor of the School of Software Engineering at South China University of Technology where she started as an Assistant Professor in 2005. She received her B.S. and Ph.D. degrees, both in Computer Science, from Sun Yat-Sen University, China, in 2000 and 2005 respectively. Her research interests can be summarized as developing effective and efficient data analysis techniques for complex data and the related applications.