Distributed Training of Structured SVM

Ching-pei Lee*
University of Wisconsin-Madison
ching-pei@cs.wisc.edu

Kai-Wei Chang*
Microsoft Research
kw@kwchang.net

Shyam Upadhyay
University of Illinois at Urbana-Champaign
upadhya3@illinois.edu

Dan Roth
University of Illinois at Urbana-Champaign
danr@illinois.edu

Abstract

Training structured prediction models is time-consuming. However, most existing approaches only use a single machine, thus, the advantage of computing power and the capacity for larger data sets of multiple machines have not been exploited. In this work, we propose an efficient algorithm for distributedly training structured support vector machines based on a distributed block-coordinate descent method. Both theoretical and experimental results indicate that our method is efficient.

1 Introduction

Many tasks in natural language processing and computer vision can be formulated as structured prediction problems, where the goal is to assign values to mutually dependent variables. The inter-dependencies constitute the "structure". To fully exploit the rich representation of the structures, it is essential to use large amount of data. However, in practice, only a limited amount of data can be used to train a structured model because most current approaches for structured learning are confined to a single machine, which imposes a limit on memory and disk capacity. For linear classification, this problem has been addressed by distributed training algorithms (see, e.g., [20, 8, 7, 18, 1]). However, there is little work on developing distributed algorithms for general structured learning.

Moreover, most existing distributed training algorithms for linear classification rely on certain properties of the objective function (e.g., differentiability). However, directly applying these methods to structured learning results in inferior convergence rates. For example, dissolve-struct[1] uses the framework in [5] for structured SVM, but this leads to a convergence rate that is only sublinear.

There are several challenges in distributed structured learning. First, the features vectors, which extracted from both the input and the output structures, are often generated on-the-fly during the training process. Synchronizing their indices across different machines may introduce additional overhead. Second, the training time of an learning algorithm consists of three parts: 1) communication, 2) inference, and 3) learning. It is important to balance these three factors. This is in contrast to linear classification, where communication is often the only bottleneck.

In this work, we address these challenges and extend the recently proposed distributed box-constrained quadratic optimization algorithm (BQO) [7] for structured support vector machines (SSVM) [16, 14]. We show that the global linear convergence rate $O(\log(1/\epsilon))$ can be obtained, even if the objective function of SSVM is non-smooth. This result is substantial, because reducing the outer iterations saves the time taken to solve the costly sub-problems. Moreover, the per-machine local sub-problems in BQO can be formed as small SSVM problems, which can be effi-

[*] Most parts of this work was done when the authors were at University of Illinois.

[1] http://dalab.github.io/dissolve-struct/.

ciently solved by off-the-shelf solvers. This enables us to leverage the well-studied single-machine structured learning methods such as the dual coordinate decent algorithm [4]. Experiments show that our algorithm is efficient and is therefore suitable for training large-scale structured models.

Existing Works. A distributed structured Perceptron algorithm using the map-reduce framework is proposed in [11]. A structured Perceptorn algorithm with mini-batch updates is discussed in [19]. However, it is unclear how to extend their algorithm on a multi-core machine to a distributed setting. When the inference problem is formulated as a factor graph, [9, 12] proposed to split the graph-based optimization problem into sub-problems, where each problem deals with a sub-graph. Then each machine solves a sub-problem in parallel and communicates with each other to enforce consistency. The convergence rate of this type of approaches is unclear. Moreover, our approach distributes instances instead of sub-graphs and is more suitable for problems with unfactorable structures and/or many instances (e.g., parsing, sequence tagging, and alignment). A simple distributed implementation of cutting plane method[2] is also available. They solve the inference problems in parallel and use one machine to learn the model. This type of approaches requires many outer iterations, and they are empirically slow even in a single machine multi-core setting (see [3]).

2 Structured Support Vector Machine

Given a set of observations $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{l}$, where $\boldsymbol{x}_i \in \mathcal{X}$ are instances with the corresponding annotated structure $\boldsymbol{y}_i \in \mathcal{Y}_i$, and $\mathcal{Y}_i$ is the set of all feasible structures for $\boldsymbol{x}_i$, SSVM solves

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \quad (1/2)\boldsymbol{w}^T\boldsymbol{w}+C\sum_{i=1}^{l}\ell(\xi_i) \qquad \text{s.t.} \qquad \boldsymbol{w}^T\phi(\boldsymbol{y},\boldsymbol{y}_i,\boldsymbol{x}_i) \geq \Delta(\boldsymbol{y}_i,\boldsymbol{y})-\xi_i, \forall i, \forall \boldsymbol{y} \in \mathcal{Y}_i, \tag{1}$$

where $C > 0$ is a predefined parameter. $\phi(\boldsymbol{y},\boldsymbol{y}_i,\boldsymbol{x}_i) = \Phi(\boldsymbol{x}_i,\boldsymbol{y}_i) - \Phi(\boldsymbol{x}_i,\boldsymbol{y})$, and $\Phi(\boldsymbol{x},\boldsymbol{y})$ is the generated feature vector depending on both the input $\boldsymbol{x}$ and the structure $\boldsymbol{y}$. $\ell(\xi)$ is the loss term to be minimized, and the loss function $\Delta(\boldsymbol{y},\boldsymbol{y}_i) \geq 0$ is a metric that represents the distance between structures. In this paper, we consider the L2-loss, $\ell(x) = x^2$.[3]

We consider solving Eq. (1) in its dual form. Let $\boldsymbol{\alpha}$ be the vector of the dual variables with dimension $\prod |\mathcal{Y}_i|$, $\otimes$ be the Kronecker product, and $\mathbf{e}$ be the vector of ones, the dual of (1) can be written as,

$$\begin{aligned} \min_{\boldsymbol{\alpha}\geq\mathbf{0}} \quad & f(\boldsymbol{\alpha}) \equiv (1/2)\boldsymbol{\alpha}^T\left(Q + A/2C\right)\boldsymbol{\alpha} - \boldsymbol{v}^T\boldsymbol{\alpha}, \\ Q_{(i,\boldsymbol{y}_1),(j,\boldsymbol{y}_2)} &= \phi(\boldsymbol{y}_1,\boldsymbol{y}_i,\boldsymbol{x}_i)^T\phi(\boldsymbol{y}_2,\boldsymbol{y}_j,\boldsymbol{x}_j), \forall 1 \leq i,j \leq l, \forall \boldsymbol{y}_1 \in \mathcal{Y}_i, \forall \boldsymbol{y}_2 \in \mathcal{Y}_j, \\ A &= (I\otimes\mathbf{e})^T(I\otimes\mathbf{e}), \qquad v_{(i,\boldsymbol{y})} = \Delta(\boldsymbol{y}_i,\boldsymbol{y}), \forall 1 \leq i \leq l, \forall \boldsymbol{y} \in \mathcal{Y}_i. \end{aligned} \tag{2}$$

From the KKT conditions, the respective optimal solutions $\boldsymbol{w}^*$ and $\boldsymbol{\alpha}^*$ to eq. (1) and eq. (2) satisfy $\boldsymbol{w}^* = \sum_{i,\boldsymbol{y}} \alpha^*_{i,\boldsymbol{y}}\phi(\boldsymbol{y},\boldsymbol{y}_i,\boldsymbol{x}_i)$. For the ease of computation, we maintain the relationship between $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ during the optimization process, and treat $\boldsymbol{w}$ as a temporary vector.

The key challenge of solving eq. (2) is that for most applications, the size of $\mathcal{Y}_i$ and thus the dimension of $\boldsymbol{\alpha}$ is exponentially large (with respect to the length of $\boldsymbol{x}_i$), so optimizing over all variables is unrealistic. Efficient dual methods [4] maintain a small working set of dual variables to be optimized such that the remaining variables are fixed to be zero. These methods then iteratively enlarge the working set until the problem is well-optimized.[4] The working set is selected using the sub-gradient of (1) with respect the current iterate. Specifically, for each training instance $\boldsymbol{x}_i$, we add the dual variable $\alpha_{i,\hat{\boldsymbol{y}}}$ corresponds to the structure $\hat{\boldsymbol{y}}$ into the working set, where

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}\in\mathcal{Y}_i} \quad \boldsymbol{w}^T\phi(\boldsymbol{y},\boldsymbol{y}_i,\boldsymbol{x}_i) - \Delta(\boldsymbol{y}_i,\boldsymbol{y}). \tag{3}$$

Once $\boldsymbol{\alpha}$ is updated, we update $\boldsymbol{w}$ accordingly. We call the step of computing eq. (3) “inference”, and call the part of optimizing eq. (2) over a fixed working set “learning”. When training SSVM distributedly, the learning step involves communication across machines. Therefore, inference and learning steps are both expensive. In the next section, we propose an algorithm that ensures fewer rounds of both parts.

[2] `http://alexander-schwing.de.`

[3] The dual form of L1-loss SSVM has an additional linear constraint, which can be viewed as a polyhedron. Thus the algorithm is still applicable and the convergence rate analysis technique is still valid.

[4] This approach is related to applying the cutting-plane methods to solve the primal problem (1) [16, 6].

Algorithm 1: A box-constrained quadratic optimization algorithm for solving (1)

```
1. $\boldsymbol{w} \leftarrow \mathbf{0}, \boldsymbol{\alpha} \leftarrow \mathbf{0}$.
2. For $t = 0, 1, \ldots$    (outer iteration)
   2.1. Use the current $\boldsymbol{w}$ to solve (4) to get $\boldsymbol{d}$ distributedly in $K$ machines.
   2.2. Use allreduce to obtain $\Delta\boldsymbol{w}$ in eq. (5).
   2.3. Compute $\eta$ by eq. (6) with another $O(1)$ communication.
   2.4. $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \eta\boldsymbol{d}$; $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\Delta\boldsymbol{w}$.
```

3 Distributed Box-Constrained Quadratic Optimization for SSVM

We split the training data into K disjoint parts, and store them in K machines. Eq. (2) is a quadratic box-constrained optimization problem; therefore, we apply the framework in [7]. At each iteration, given the current $\boldsymbol{\alpha}$ and a symmetric positive definite H, we solve

$$\boldsymbol{d} = \arg\min\nolimits_{\boldsymbol{d}:\boldsymbol{\alpha}+\boldsymbol{d}\geq\mathbf{0}} \quad g_H(\boldsymbol{d}) \equiv \nabla f(\boldsymbol{\alpha})^T\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T H\boldsymbol{d}. \tag{4}$$

We then conduct a line search to decide a suitable step size η and update $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \eta\boldsymbol{d}$. The detailed description is in Algorithm 1. Here, we consider $H \equiv \theta\bar{Q} + \frac{1}{2C}A + \lambda I$, where $\lambda > 0$ is a small constant to ensure $H \succ 0$, $\theta > 0$ can be tuned to decide how conservative the updates are, and

$$\bar{Q}_{(i,\boldsymbol{y}_1),(j,\boldsymbol{y}_2)} = \begin{cases} 0 & \text{if } i, j \text{ are not in the same partition,} \\ \phi(\boldsymbol{y}_1, \boldsymbol{y}_i, \boldsymbol{x}_i)^T\phi(\boldsymbol{y}_2, \boldsymbol{y}_j, \boldsymbol{x}_j) & \text{otherwise.} \end{cases}$$

The choice of H is based on two factors: 1) To converge fast, H should be an approximation of the real Hessian; 2) To solve eq. (4) without incurring communication cost across different machines, H should be decomposable to sub-matrices, where each sub-matrix uses information from data stored on one machine. Our design of H enables eq. (4) to be split into K sub-problems and solved locally. Each sub-problem can be rewritten as a SSVM dual problem. Thus, one can adopt any single-machine SSVM solver (e.g., [6, 16, 15, 4, 13]) to solve it. After (4) is solved, we compute

$$\Delta\boldsymbol{w} \equiv \sum\nolimits_{i,\boldsymbol{y}} \boldsymbol{d}_{i,\boldsymbol{y}}\phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i) \tag{5}$$

by an *allreduce* operation that communicates information between machines. This information also synchronizes the model for conducting inferences to enlarge the working set. Using $\Delta\boldsymbol{w}$, an exact line search for deciding the optimal step size η^* can be conducted.

$$\frac{\partial f(\boldsymbol{\alpha} + \eta\boldsymbol{d})}{\partial\eta} = 0 \Rightarrow \eta^* = \frac{-\nabla f(\boldsymbol{\alpha})^T\boldsymbol{d}}{\boldsymbol{d}^T(Q + A/2C)\boldsymbol{d}} = -\frac{\boldsymbol{w}^T\Delta\boldsymbol{w} + \boldsymbol{\alpha}^T(A/2C)\boldsymbol{d} - \boldsymbol{v}^T\boldsymbol{d}}{\Delta\boldsymbol{w}^T\Delta\boldsymbol{w} + \boldsymbol{d}^T(A/2C)\boldsymbol{d}}.$$

To ensure feasibility, we take the final step size η to be

$$\eta = \min(\max\{\eta' \mid \boldsymbol{\alpha} + \eta'\boldsymbol{d} \geq \mathbf{0}\}, \eta^*). \tag{6}$$

Following the analysis in [7], we can show the following convergence result for Algorithms 1.

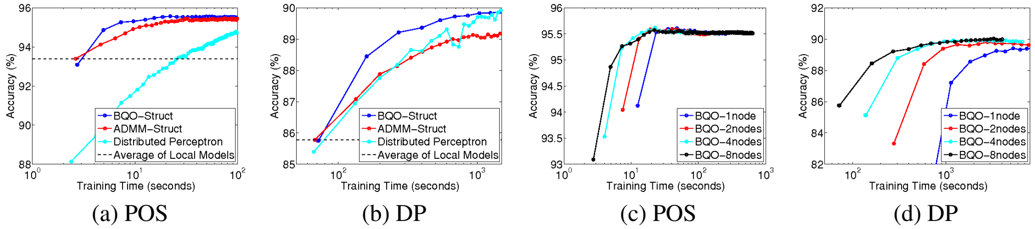
Theorem 1. *Algorithm 1 has global linear convergence when the exact solution of* (4) *is obtained at each iteration and* $H \succ 0$.

In practice, obtaining the exact solution of (4) is time-consuming. We show that global linear convergence still holds when (4) is solved approximately.

Corollary 1. *Let* $\boldsymbol{d}^*$ *be the optimal solution of* (4). *If for some constant* $\gamma \in [0, 1)$ *and for all t, the update direction* $\boldsymbol{d}$ *satisfies* $\gamma|g_H(\boldsymbol{d}^*)| \leq |g_H(\boldsymbol{d})|$ *with* $H \succ 0$, *then Algorithm 1 converges with a global linear rate.*

Since γ is arbitrary, for any sub-problem solver that strictly decreases the function value, we can easily obtain a value of $\gamma < 1$.

The communication step in eq. (5) requires machines to communicate a vector of $O(n)$. The actual cost of this communication depends on the network setting and usually grows with K. We note that solving (4) approximately results in more iterations and thus more rounds of communication, but requires fewer inference calls. Thus this is a trade-off between communication and inference. For many applications, inference is much more expensive than communication, thus the balance between these two factors is worth studying empirically.



(a) POS (b) DP (c) POS (d) DP

Figure 1: 1a and 1b: Comparison between different algorithms using eight nodes. 1c and 1d: Performance of BQO-STRUCT using different number of machines. Training time is in *log scale*.

Model Consistency Unlike binary classification, while learning a structured model, features are usually generated on-the-fly because the feature set depends on the structures the solver has seen so far. If each machine maintains its own feature mapping, the feature indices will be inconsistent across machines. One potential solution is to synchronize the feature mappings at each round. However, this approach incurs a huge communication overhead. To tackle this issue, we adapt a feature hashing strategy in [17]. We map the features into integer values in $[0, 2^d), d \in \mathcal{N}$ by a unique hashing function and use them as new feature indices, such that the size of the weight vector is at most 2^d. The input to this hashing function can be any object, such as an integer or a string. This strategy has been used in distributed environments [1, 9] for dimension reduction and fast look-up. Here, as argued before, this techniques is crucial and efficient for distributed structured learning.

4 Experiments

We perform experiments on part-of-speech tagging (POS) and dependency parsing (DP). For both tasks, we use the Wall Street Journal portion of the Penn Treebank [10] with the standard split for training (section 02-21) and test (section 23). For both tasks, we set $C = 0.1$ for SSVM and compare the following algorithms using eight nodes in a local cluster.

1. BQO-STRUCT: the algorithm we proposed in Section 3. We set θ to be K.
2. ADMM-STRUCT: the alternating directions method of multiplier [2].
3. DISTRIBUTED PERCEPTRON: a parallel structured Perceptron algorithm described in [11].
4. Simple average: Each machine trains a separate model using the local data. The final model is obtained by averaging all local models.

The sub-problems in ADMM-STRUCT and BQO-STRUCT are solved by the dual coordinate descent solver proposed in [4], which is shown to be empirically faster than other existing methods. To have a fair comparison, we use the same setting for solving sub-problems when possible.

Because different methods solve different objectives, we compare the test performance along training time. Figure 1 shows the results. BQO-STRUCT performs the best in both tasks, confirming its fast theoretical convergence rate. We further investigate the speedup of BQO-STRUCT in Figures 1c-1d. This also serves as a comparison between our distributed algorithm and the state-of-the-art single-machine SSVM solver. For the time-consuming task DP, the speedup is significant because a large portion of the training time is spent on inference. Parallelizing this part can achieve nearly linear speedup. While for POS, because the training time using a single machine is already fast enough, using multiple machines does not improve the training time much.

Overall, this work addresses the challenge of training structured SVM problems in a distributed setting and proposes an algorithm with fast convergence rate and good empirical performance. We hope this work will inspire more applications of structured learning with large volume of training data to improve the performance on structured learning tasks.

This research was supported by the Multimodal Information Access & Synthesis Center at UIUC, part of CCICADA, a DHS Science and Technology Center of Excellence and by DARPA under agreement number FA8750-13-2-0008. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U. S. Government.

References

[1] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, 2014.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[3] K.-W. Chang, V. Srikumar, and D. Roth. Multi-core structural SVM training. In *ECML*, 2013.

[4] M.-W. Chang and W.-T. Yih. Dual coordinate descent algorithms for efficient large margin structural learning. *Transactions of the Association for Computational Linguistics*, 2013.

[5] M. Jaggi, V. Smith, M. Takáč, J. Terhorst, T. Hofmann, and M. I. Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems 27*. 2014.

[6] T. Joachims, T. Finley, and C.-N. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 2009.

[7] C.-P. Lee and D. Roth. Distributed box-constrained quadratic optimization for dual linear SVM. In *ICML*, 2015.

[8] C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin. Large-scale logistic regression and linear support vector machines using Spark. In *Proceedings of the IEEE International Conference on Big Data*, pages 519–528, 2014.

[9] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8), 2012.

[10] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*.

[11] R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured Perceptron. In *ACL*, 2010.

[12] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun. Efficient structured prediction with latent variables for general graphical models. In *ICML*, 2012.

[13] S. K. Shevade, B. P., S. Sundararajan, and S. S. Keerthi. A sequential dual method for structural SVMs. In *SDM*, 2011.

[14] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems 16*. 2004.

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

[16] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 2005.

[17] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *ICML*, 2009.

[18] C. Zhang, H. Lee, and K. G. Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *AISTATS*, 2012.

[19] K. Zhao and L. Huang. Minibatch and parallelization for online large margin structured learning. In *NAACL*, pages 370–379, 2013.

[20] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. Distributed Newton method for regularized logistic regression. In *PAKDD*, 2015.