Large-Scale Graph-based Transductive Inference

Amarnag Subramanya[*]
University of Washington, Seattle.
Department of Electrical Engineering
asubram@u.washington.edu

Jeff Bilmes
University of Washington, Seattle.
Department of Electrical Engineering
bilmes@ee.washington.edu

Abstract

We consider the issue of scalability of graph-based semi-supervised learning (SSL) algorithms. In this context, we propose a fast graph node ordering algorithm that improves (parallel) spatial locality by being cache cognizant. This approach allows for a near linear speedup on a shared-memory parallel machine to be achievable, and thus means that graph-based SSL can scale to very large data sets. We use the above algorithm in a multi-threaded multi-core implementation to solve a SSL problem on a 120 million node graph in a reasonable amount of time.

1 Introduction

In *semi-supervised Learning* (SSL) small amounts of labeled data are used with large amounts of unlabeled data for training classifiers. For a survey of SSL algorithms, see [1, 2]. In many applications, such as speech recognition, while annotating training data is time-consuming, tedious and error-prone, large amounts of unlabeled data is obtained very easily. As a result in SSL we have access to large amounts of unlabeled data and thus the issue of scalability is a critical issue for wide deployment of SSL. In this paper we focus on graph-based SSL algorithms [1] where the assumptions is that the labeled and unlabeled samples are embedded within a low-dimensional manifold expressed by a graph. In [3, 4] we proposed a new algorithm for graph-based SSL called *measure propagation* (MP) which addresses some of the drawbacks of the current state-of-the-art.

The problem of scalability has not received much attention in SSL (in particular graph-based SSL). [5] suggests an algorithm for improving the induction speed in the case of graph-based algorithms. [6] solves a graph transduction problem with 650,000 samples. To the best of our knowledge, the largest graph-based problem solved to date had about 900,000 samples (which includes both labeled and unlabeled data) [7]. Clearly, this is a fraction of the amount of unlabeled data at our disposal. For example, on the Internet alone, we create 1.6 billion blog posts, 60 billion emails, 2 million photos and 200,000 videos *every day* [8]. In this paper, we use the standard phone classification problem to show how graph-based algorithms (in particular MP) may be scaled to very large problems. We note that our approach would work for any iterative graph-based algorithm (such as label propagation). We believe that speech recognition is an ideal application for SSL and in particular graph-based SSL on account of two reasons – (a) human speech is produced by a small number of articulators and thus amenable to representation by a low-dimensional manifold [9], and (b) annotating speech data is time-consuming, tedious, costly, and often error prone.

[*]Now at Google Research, Google Inc. asubram@google.com

2 Graph-based SSL

Given a graph $\mathcal{G} = (V, E)$ over m samples of which the first l samples are labeled and remaining samples are unlabeled, MP is based on minimizing the following objective

$$\mathcal{C}(\mathrm{p}, \mathrm{q}) = \sum_{i=1}^{l} D_{KL}\big(r_i || q_i\big) + \mu \sum_{i=1}^{m} \sum_{j \in \mathcal{N}'(i)} w'_{ij} D_{KL}\big(p_i || q_j\big) - \nu \sum_{i=1}^{m} H(p_i).$$

where $H(p)$ ad $D_{KL}(p||q)$ are the Shannon entropy and Kullback-Leibler divergence respectively. r_i, p_i and q_i are all multinomial distributions (in general they can be any unsigned measure, see [4]). Here $r_i, i \in \{1, \cdots, l\}$ is an encoding of the labels, $p_i(y)$ and $q_i(y)$ represent the probability that vertex i belong to class y. The q_i's are introduced to make the objective amenable for optimization using alternating minimization (AM). $\mu, \nu > 0$ are hyperparameters whose choice we discuss in section 4. Finally, $w'_{ij} = \Big[\mathbf{W}'\Big]_{ij}$ and $\mathbf{W}' = \mathbf{W} + \alpha \mathbf{I}_n$, $\mathcal{N}'(i) = \{\{i\} \cup \mathcal{N}(i)\}$ and $\alpha \geq 0$. α, which is a hyper-parameter, plays an important role in ensuring that p_i and q_i are close $\forall\ i$ at convergence. Our results from [3, 4] suggest that setting $\alpha = 2$ ensures that $p^*_i(y) = q^*_i(y),\ \forall\ i, y$ (assuming p^* and q^* are the optimal solutions). The first term in $\mathcal{C}$ encourages q_i for the labeled vertices to be close to the labels, r_i, the last term encourages higher entropy p_i's. The second term, in addition to acting as a graph regularizer, also acts as glue between the p's and q's. It is also possible to show (see [4]) that the update equations for solving $\mathcal{C}$ are given by

$$p_i^{(n)}(y) = \frac{\exp\{\frac{\mu}{\gamma_i} \sum_j w'_{ij} \log q_j^{(n-1)}(y)\}}{\sum_y \exp\{\frac{\mu}{\gamma_i} \sum_j w'_{ij} \log q_j^{(n-1)}(y)\}} \quad \text{and } q_i^{(n)}(y) = \frac{r_i(y)\delta(i \leq l) + \mu \sum_j w'_{ji} p_j^{(n)}(y)}{\delta(i \leq l) + \mu \sum_j w'_{ji}}$$

where $\gamma_i = \nu + \mu \sum_j w'_{ij}$. For more information on this objective see [3, 4].

3 Parallelism and Scalability to Large Datasets

3.1 Multiple Threads with Shared Memory

From the above update equations it can be seen that one set of measures is held fixed while the other set is updated without any required communication amongst set members, so there is no write contention. This immediately yields a $T \geq 1$-threaded implementation where the graph is evenly T-partitioned and each thread operates over only a size $m/T = (l+u)/T$ subset of the graph nodes. We first attempted a naive multi-threaded implementation of MP and measured its performance on a graph with 1.4 million vertices. We ran a timing test on a 16 core symmetric multiprocessor with 128GB of RAM, each core operating at 1.6GHz, and no more than one thread per core. The number of threads T was varied within the set $\{1, \ldots, 16\}$. In each case running 3 iterations of AM (i.e., 3 each of p and q updates). Each experiment was repeated 10 times, and we measured the minimum CPU time over these 10 runs (total CPU time only was taken into account). The speedup for T threads is typically defined as the ratio of time taken for single thread to time taken for T threads. The solid (black) line in figure 1(a) represents the ideal case (a linear speedup), i.e., when using T threads results in a speedup of T. The pointed (green) line shows the actual speedup of the above procedure, typically less than ideal due to inter-process communication and poor shared L1 and/or L2 microprocessor cache interaction and/or poor translation lookaside buffer (TLB) use. When $T \leq 4$, the speedup (green) is close to ideal, but for increasing T the algorithm diminishes away from the ideal case.

Our contention is that the sub-linear speedup is due to the poor machine cognizance of the algorithm. At a given point in time, suppose thread $t \in \{1, \ldots, T\}$ is operating on node i_t. The collective set of neighbors that are being used by these T threads are $\{\cup_{t=1}^{T} \mathcal{N}(i_t)\}$ and this, along with nodes $\cup_{t=1}^{T}\{i_t\}$ (and all memory for the associated measures), constitute the current *working set*. The working set should be made as small as possible to increase the chance it will fit in the microprocessor caches, but this becomes decreasingly likely as T increases since the working set is monotonically increasing with T. Our goal, therefore, is for the nodes that are being simultaneously operated on to have a large amount of neighbor overlap thus minimizing the working set size. This can be cast as an optimization problem in a number of ways: the goal could be to find a partition $(V_1, V_2, \ldots, V_{m/T})$ of V that minimizes $\max_{j \in \{1, \ldots, m/T\}} |\cup_{v \in V_j} \mathcal{N}(v)|$. With such a partition, we

Algorithm 1 Graph Ordering Algorithm

Select an arbitrary node v.
while there are any unselected nodes remaining **do**
  Let $\mathcal{N}(v)$ be the set of neighbors, and $\mathcal{N}^2(v)$ be the set of neighbors' neighbors, of v.
  Select a currently unselected $v' \in \mathcal{N}^2(v)$ such that $|\mathcal{N}(v) \cap \mathcal{N}(v')|$ is maximized. If the intersection is empty, select an arbitrary unselected v'.
  $v \leftarrow v'$.
end while

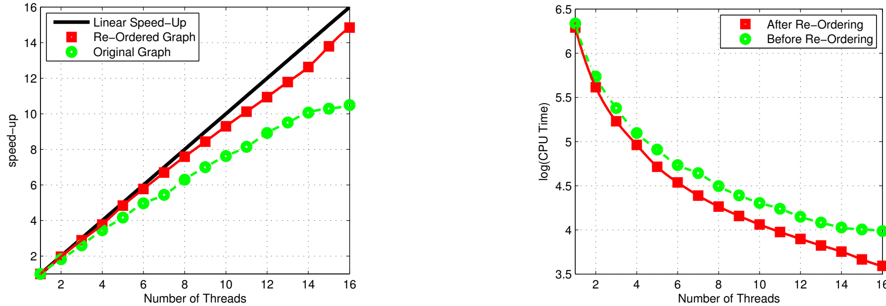

Figure 1: (a) speedup vs. number of threads for the TIMIT graph (see section 4). The process was run on a 128GB, 16 core machine with each core at 1.6GHz. (b) The *actual* CPU times in seconds on a *log scale* vs. number of threads for with and without ordering cases.

may also order the subsets so that the neighbors of V_i would have maximal overlap with the neighbors of V_{i+1}. We then schedule the T nodes in V_j to run simultaneously, and schedule the V_j sets successively.

Of course, the time to produce such a partition cannot dominate the time to run the algorithm itself. Therefore, we propose a simple fast node ordering procedure (Algorithm 1) that can be run once before the parallelization begins. The algorithm orders the nodes such that successive nodes are likely to have a high amount of neighbor overlap with each other and, by transitivity, with nearby nodes in the ordering. It does this by, given a node v, choosing another node v' (from amongst v's neighbors' neighbors, meaning the neighbors of v's neighbors) that has the highest neighbor overlap. We need not search all V nodes for this, since anything other than v's neighbors' neighbors has no overlap with the neighbors of v. Given such an ordering, the tth thread operates on nodes $\{t, t + m/T, t+2m/T, \ldots\}$. If the threads proceed perfectly synchronously (which we do not enforce) the set of nodes being processed at any time instant are $\{1 + jm/T, 2 + jm/T, \ldots, T + jm/T\}$. This assignment is beneficial not only for maximizing the set of neighbors being simultaneously used, but also for successive chunks of T nodes since once a chunk of T nodes have been processed, it is likely that many of the neighbors of the next chunk of T nodes will already have been pre-fetched into the caches. With the graph represented as an adjacency list, and sets of neighbor indices sorted, our algorithm is $O(mk^3)$ in time and linear in memory since the intersection between two sorted lists may be computed in $O(k)$ time. This is typically even better than $O(m \log m)$ since $k^3 < \log m$ for large m.

We ordered graph nodes obtained from the TIMIT corpus (a standard speech corpus, see [10, 4]) and ran timing tests as explained above. In this case, one 25ms window of speech corresponded to one graph node, where successive windows have a 15ms overlap (a standard value in the speech recognition literature). The time required for node ordering **is** also counted in each timing run — specifically, all timing numbers reported when the graph ordering algorithm is run includes the time to perform the graph ordering algorithm. Results are shown in figure 1(a) (pointed red line) where the results are much closer to ideal, and there are no obvious diminishing returns like in the unordered case. Running times are given in figure 1(b). Moreover, the ordered case showed better performance even for a single thread $T = 1$ (CPU time of 539s vs. 565s for ordered vs. unordered respectively, on 3 iterations of AM).

We conclude this section by noting that (a) re-ordering may be considered a pre-processing (offline) step, (b) the SQ-Loss algorithm may also be implemented in a multi-threaded manner and this is supported by our implementation, (c) our re-ordering algorithm is general and fast and can be used for any graph-based algorithm where the iterative updates for a given node are a function of its neighbors (i.e., the updates are harmonic w.r.t. the graph [11]), and (d) while the focus here was on parallelization across different processors on a symmetric multiprocessor, this would also apply for distributed processing across a network with a shared disk.

4 Results on Large Datasets

In [10, 4] we showed how the above reordering algorithm can be used with a multi-threaded implementation of MP to solve a SSL problem with 120 million samples on a single machine (16 core symmetric multiprocessor with 128GB of RAM). In addition, and as a result, we also showed that MP outperforms previously proposed approaches (specifically label propagation, LP) for which it was feasible to run on such a large data set. While the objective of that paper was more to show that MP outperforms LP, we note that the graph ordering algorithm was used both for MP and for LP and achieved comparable improvements in performance. We therefore content that our procedure provides a practical scheme for ordering the nodes in a variety of graph-based message-passing algorithms, including algorithms that have a very different purpose such as belief propagation for solving inference in graphical models. We note also that ours was the first attempt at solving a SSL problem of this scale and magnitude.

References

[1] O. Chapelle, B. Scholkopf, and A. Zien, *Semi-Supervised Learning*. MIT Press, 2007.

[2] X. Zhu, "Semi-supervised learning literature survey," tech. rep., Computer Sciences, University of Wisconsin-Madison, 2005.

[3] A. Subramanya and J. Bilmes, "Soft-supervised text classification," in *EMNLP*, 2008.

[4] A. Subramanya and J. Bilmes, "Entropic graph regularization in non-parametric semi-supervised classification," in *NIPS*, 2009.

[5] O. Delalleau, Y. Bengio, and N. L. Roux, "Efficient non-parametric function induction in semi-supervised learning," in *Proc. of the Conference on Artificial Intelligence and Statistics (AISTATS)*, 2005.

[6] M. Karlen, J. Weston, A. Erkan, and R. Collobert, "Large scale manifold transduction," in *International Conference on Machine Learning, ICML*, 2008.

[7] I. W. Tsang and J. T. Kwok, "Large-scale sparsified manifold regularization," in *Advances in Neural Information Processing Systems (NIPS) 19*, 2006.

[8] A. Tomkins, "Keynote speech." CIKM Workshop on Search and Social Media, 2008.

[9] A. Jansen and P. Niyogi, "Semi-supervised learning of speech sounds," in *Interspeech*, 2007.

[10] A. Subramanya and J. Bilmes, "The semi-supervised switchboard transcription project," in *Interspeech*, 2009.

[11] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proc. of the International Conference on Machine Learning (ICML)*, 2003.