Scalable optimization methods for Distributed Machine Learning

Parameswaran Raman
University of California, Santa Cruz

1 Introduction

As data grows, there is a need to develop scalable, efficient and most importantly distributed machine learning algorithms. Distributed algorithms fall into two main categories - (a) horizontal partitioning, where the data is distributed across multiple slaves. The main drawback of this strategy is that the model parameters need to be replicated on every machine. This is problematic when the number of classes, and consequently the number of parameters is very large, and hence cannot fit in a single machine. The other strategy is (b) vertical partitioning, where the model parameters are partitioned. However, here the data needs to be replicated on each machine, thus failing to scale to massive datasets.

My goal is to get the best of both worlds by partitioning both - the data as well as the model parameters, thus opening the door to training more sophisticated models on massive datasets.

Typical approaches to scaling algorithms include leveraging directly observable structure in the data such as sparsity, hierarchical relationships among the labels, low-rank conditions on the data matrix and so on. My research on the other hand, focusses on reformulating the problem to yield a structure that allow both horizontal and vertical partitioning simultaneously. One such structure we identified in many machine learning problems is *double-separability* and my thesis work focusses on showing how it can be used to develop scalable and efficient algorithms for a variety of problems.

Such large scale problems arise in a number of real-world tasks. For instance,

- *Learning to Rank*, where the number of items to be ranked for a given user is in the order of millions
- *Matrix Factorization*, where the matrix of users and items contains millions of non-zero entries
- *Multi-class* classification where the number of classes is in the order of millions
- *Multi-label* classification where the number of labels is in the order of millions

In this direction, I have worked on a ranking algorithm - *RoBiRank* (Ranking via Robust Binary Classification) [8], where we develop a ranking algorithm showing close connections to Robust Binary Classification and NDCG, the popular evaluation metric used for ranking. In addition, we show how to scale our algorithm to massive datasets.

In my second project (on-going work), I am working on *DS-MLR (Doubly Separable MLR)* a stochastic gradient descent based optimization method for scaling up multinomial logistic regression problems to very large number of classes.

2 Ranking via Robust Binary Classification (RoBiRank)

First, on the Learning to Rank task, we motivate the construction of our ranking algorithm RoBiRank and demonstrate some novel connections. Next, we show how RoBiRank can also be scaled to very large datasets using the Latent Collaborative Retrieval task.

2.1 Learning to Rank

Learning to rank is a problem of ordering a set of items according to their relevances to a given context. Examples could include ranking a set of movies for a given user on netflix or ordering a set of documents for a given search query on the internet. One popular way to approach the learning to rank problem is using the pairwise method where ranking is transformed into pairwise classification. A classifier for classifying the ranking orders of item

pairs is created and is employed in the ranking of items. The objective function for such a model can be expressed as:

$$L(\omega) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}_x} r_{xy} \sum_{y' \in \mathcal{Y}_x, y' \neq y} \sigma\left(f_\omega(x,y) - f_\omega(x,y')\right) \tag{1}$$

where,

- $\mathcal{X}$ = set of users, $\mathcal{Y}$ = set of items, r_{xy} = rating user x gave to item y
- $\phi(x,y) \in \mathbb{R}^d$: extracted feature between x and y, $\omega \in \mathbb{R}^d$: model parameter
- $f_\omega(x,y) := \langle \phi(x,y), \omega \rangle$: the score model assigns to item y for user x
- $\sigma(\cdot)$:= any convex loss function (e.g.: logistic, hinge)

However, it is well known that convex loss functions are sensitive to outliers [5]. Therefore, if used in a ranking model such as above, they cause the ranker to pay more attention on the bad items affecting the overall performance.

Connection to Robust Binary Classification: To tackle the above issue, we propose bending the convex losses by using special transformation functions that are popular in the Robust Binary Classification literature [8]. We realize that one such transformation function also preserves the gradient information and hence we use it to our advantage for optimization. Incorporating the robust transformation function $\rho(\cdot)$ in our original ranking loss in equation (1), we get:

$$L(\omega) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}_x} r_{xy} \cdot \rho\left(\sum_{y' \in \mathcal{Y}_x, y' \neq y} \sigma\left(f_\omega(x,y) - f_\omega(x,y')\right)\right) \tag{2}$$

Connection to Discounted Cumulative Gain: *Discounted Cumulative Gain (DCG)* is a standard evaluation measure used in learning to rank systems and it measures the usefulness, or gain, of an item based on its position in the result list. The gain of each result is discounted at lower ranks logarithmically. Mathematically, it can be expressed as:

$$\text{DCG}(\omega) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}_x} \frac{r_{xy}}{\log_2\left(rank_\omega(x,y) + 2\right)},$$

Interestingly, we observe that maximizing DCG is equivalent to minimizing our robust ranking loss function $L(\omega)$ in equation (2). Therefore, our algorithm directly optimizes for the best DCG.

Using such a ranking loss helps us in two ways - (1) our algorithm learns to give up on bad items, (2) focusses on quality at the top of the ranking list. We carried out extensive experiments on small to medium datasets comparing several widely used ranking algorithms and our results as shown in figure (1) indicate that the accuracy (NDCG) of our algorithm RoBiRank is higher at the top of the list, which is a desirable quality to have for a real-world application.

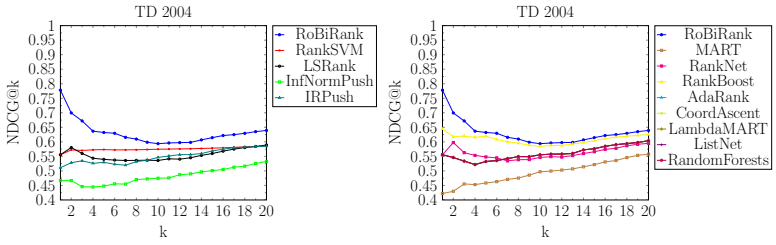

Figure 1: Results on small, medium scale data

2.2 Latent Collaborative Retrieval

Often in real-world applications, the number of users $\mathcal{X}$ or the number of items $\mathcal{Y}$ might be too large to be able to generate joint features $\phi(x,y)$ for every x and y pair. To avoid this feature engineering burden, we can make use of implicit features available in the data. For example, a lot of users on Netflix would simply watch movies without leaving an explicit rating, but by the action of watching a movie they implicitly express their preference. We can

attempt to learn the score function $f(x, y)$ without a feature vector $\phi(x, y)$ by embedding each context and item in an Euclidean latent space; specifically, we redefine the score function to be: $f(x, y) := \langle U_x, V_y \rangle$, where $U_x \in \mathbb{R}^d$ is the embedding of the context x and $V_y \in \mathbb{R}^d$ is that of the item y. Then, we can learn these embeddings by a ranking model. This approach was introduced in [6], and was called *latent collaborative retrieval*.

We specialize RoBiRank model for this task and show how to scale it to large data. The reformulated objective function becomes:

$$\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}_x} r_{xy} \cdot \rho \left(\sum_{y' \in \mathcal{Y}_x, y' \neq y} \sigma \left(\langle U_x, V_y \rangle - \langle U_x, V_{y'} \rangle \right) \right). \tag{3}$$

where,

- $U_1, U_2, \ldots, U_n \in \mathbb{R}^d$ are the user parameters (embeddings)
- $V_1, V_2, \ldots, V_m \in \mathbb{R}^d$ are the item parameters (embeddings)
- score is defined as $f_\omega(x, y) := \langle U_x, V_y \rangle$

Developing an Unbiased Stochastic Gradient Algorithm: Computing the sum over all $\mathcal{Y}_x$ in the above formulation is computationally expensive when the number of items is large. Therefore, to avoid calculating the summation over $\mathcal{Y}_x$, using the following property of $\rho(\cdot)$,

$$\rho(t) = \log_2(t+1) \leq -\log_2 \xi + \frac{\xi \cdot (t+1) - 1}{\log 2}, \quad \text{(for any } \xi > 0)$$

we *linearize* the objective function as:

$$\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}_x} r_{xy} \cdot \left[-\log_2 \xi_{xy} + \frac{\xi_{xy} \cdot \left(\sum_{y' \neq y} \sigma \left(\langle U_x, V_y \rangle - \langle U_x, V_{y'} \rangle \right) + 1 \right) - 1}{\log 2} \right], \tag{4}$$

by introducing ξ_{xy} for each x, y with $r_{xy} \neq 0$.

Now, if we uniformly sample (x, y, y') from $\{(x, y, y') : r_{xy} \neq 0\}$, we obtain an **unbiased estimator**, which allows us to take **stochastic gradient** with convergence guarantees.

Parallelization and Data Distribution Scheme: The user and item parameters are partitioned into multiple machines in such a way that the user parameters always are fixed, while the item parameters are exchanged after each epoch of optimization. Within each epoch, stratified SGD updates are done independently per machine using the block of data held by the machine [2]. This is shown in the figure (2) below.

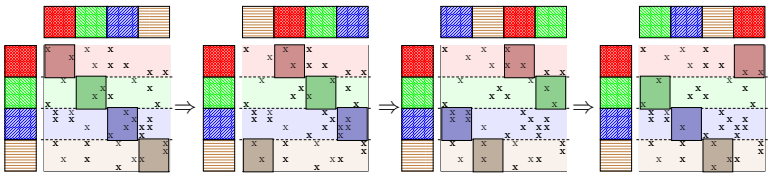

Figure 2: Stratified SGD

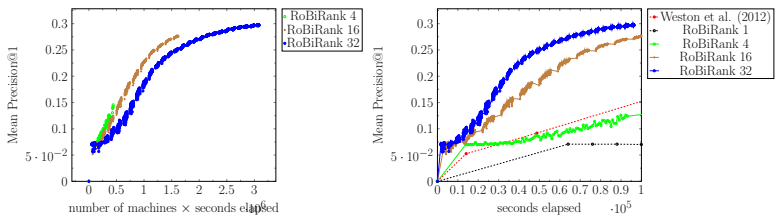

Figure 3: Large Scale Experiments

We apply our algorithm to latent collaborative retrieval task on Million Song Dataset, which consists of 1,129,318 users, 386,133 songs, and 49,824,519 records; for this task, a ranking algorithm has to optimize an objective function

that consists of 386,133 × 49,824,519 number of pairwise interactions. With the same amount of wall-clock time given to each algorithm, RoBiRank leverages parallel computing to outperform the state-of-the-art [6] with a 100% lift on the evaluation metric. As can be seen in figure (3), RoBiRank scales nicely up to 32 machines (16 cores each) exhibiting a linear scaling behavior.

3 Doubly Separable Multinomial Logistic Regression (DS-MLR)

Suppose the training data consists of N data points $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ where $x_i \in \mathbb{R}^d$ is a d-dimensional feature vector and $y_i \in \{1, 2, \ldots, K\}$ is a label associated with it; K denoting the number of class labels. The objective function of MLR can be written in a regularized risk minimization form as:

$$L(W) = \frac{\lambda}{2} \sum_{k=1}^{K} \|w_k\|^2 - \sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} w_k^T x_i + \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} exp(w_k^T x_i) \right) \tag{5}$$

where λ is the regularization parameter.

Optimizing the above objective function (5) when the number of classes K become large is extremely challenging as computing the log-sum-exp function (partition function) involves summing up over the large number of classes. In addition, the log partition function couples the class level parameters w_k inside, making it difficult to parallelize MLR vertically by optimizing each w_k independently across multiple machines. In order to tackle this, Gopal and Yang [3] propose a way to replace the log partition function by a parallelizable function. More specifically, they use the *Log-concavity bound* Bouchard [1] to upper bound the partition function. This bound is given by:

$$\log(\gamma) \leq a\gamma - \log(a) - 1, \forall \gamma, a > 0 \tag{6}$$

where a is a variational parameter. This bound is tight when $a = \frac{1}{\gamma}$. Applying this bound, the overall objective for MLR in eqn (5) can be written as:

$$L(W, A) = \frac{\lambda}{2} \sum_{k=1}^{K} \|w_k\|^2 + \sum_{i=1}^{N} \left(-\sum_{k=1}^{K} y_{ik} w_k^T x_i + a_i \sum_{k=1}^{K} exp(w_k^T x_i) - \log(a_i) - 1 \right) \tag{7}$$

The above objective although not convex, is differentiable and Gopal and Yang [3] proposed a block coordinate descent procedure for optimization. They term their method LC (for Log-Concavity) and provide a distributed implementation based on the Hadoop Map-Reduce framework. In each iteration, the class level w_k parameters are optimized in parallel in the *map* phase and the variational parameters are updated in the *reduce* phase. Firstly, the LC method requires data to be replicated across all machines while doing the optimization and hence cannot be scaled to very large datasets which do not fit on a single machine. Secondly, the LC method de-couples the class level parameters w_k thereby loosing good opportunities to better convergence. Our stochastic gradient based algorithm DS-MLR addresses these drawbacks.

Our contribution: We make use of a concept known as *Double-Separability* in our new algorithm. It turns out that several models in machine learning can be expressed in a doubly-seprable form and Yun [7] studies this in much more detail. Such objective functions can be optimized parallely in two directions. In other words, both *data partitioning* as well as *model partitioning* can be carried out simultaneously. For instance, Stochastic Gradient Descent (SGD) is known to be inherently sequential, but using the Double-Separability in our model, we are able to come up with a parallelized SGD algorithm for MLR. Double-Separability also has several systems level implications such as minimizing the number of locks needed on blocks of parameters while performing optimization and being easy to parallelize.
We can easily re-write objective (7) in a doubly-separable form as:

$$L(W, A) = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(\frac{\lambda}{2N} \|w_k\|^2 - \frac{y_{ik} w_k^T x_i}{N} + \frac{\exp(w_k^T x_i + b_i)}{N} - \frac{b_i}{KN} - \frac{1}{KN} \right) \tag{8}$$

where $b_i = \log(a_i)$

As one can see, the objective now splits as summations over the N data points and K classes making it possible to derive stochastic gradient estimates. Computing stochastic gradients $\nabla(w_k)$ and $\nabla(b_i)$ we can develop an efficient distributed SGD algorithm. Two possibilities include:

(a) Stochastic W, Stochastic A:
An instance i and a class k are randomly chosen from $\{1, 2, \ldots, N\}$ and $\{1, 2, \ldots, K\}$ respectively to update W and A. The update rules in this case look like:

$$w_k \leftarrow w_k - \eta_1 \left(\frac{\lambda}{N} w_k - \frac{y_{ik} x_i}{N} + \frac{\exp(w_k^T x_i + b_i) x_i}{N} \right)$$
$$b_i \leftarrow b_i - \eta_2 \left(\frac{\exp(w_k^T x_i + b_i)}{N} - \frac{1}{KN} \right)$$

where η_1 and η_2 are learning rates for w_k and b_i respectively.

(b) Stochastic W, Exact A:
An instance $k \leftarrow \{1, 2, \ldots, K\}$ is chosen randomly to update W, while A is updated using the closed form given by:

$$a_i = \frac{1}{\sum_{k=1}^{K} \exp(w_k^T x_i)} \tag{9}$$

To parallelize, we can use the same data-distribution scheme as used in our previous work RoBiRank to partition the work among various machines.

4 Future Directions

As next steps, I am interested in extending our DS-MLR algorithm to large scale multi-label problems and possibly exploring ways to exploit the label hierarchy in an efficient manner. Another interesting direction is scalable bayesian non-parametric models. I wish to spend some of my future efforts on methods like Variational Inference and how to scale them to massive datasets. To avoid complete pass over the data, Stochastic Variational Inference [4] has been proposed which makes use of mini-batches of data to update the local and global variational parameters. Although this is faster, there can be significant speedups using a double-stochastic structure (one coming from the mini-batches of data and another from the variational parameters). This is closely connected to our doubly-separable formulation in DS-MLR.

References

[1] Guillaume Bouchard. Efficient bounds for the softmax function, applications to inference in hybrid models. 2007.

[2] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Conference on Knowledge Discovery and Data Mining*, pages 69–77, 2011.

[3] Siddharth Gopal and Yiming Yang. Distributed training of large-scale logistic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 289–297, 2013.

[4] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347, 2013. URL `http://jmlr.org/papers/v14/hoffman13a.html`.

[5] Phil Long and Rocco Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning Journal*, 78(3):287–304, 2010.

[6] Jason Weston, Chong Wang, Ron Weiss, and Adam Berenzweig. Latent collaborative retrieval. *arXiv preprint arXiv:1206.4603*, 2012.

[7] Hyokun Yun. *Doubly Separable Models*. PhD thesis, Purdue University West Lafayette, 2014.

[8] Hyokun Yun, Parameswaran Raman, and S Vishwanathan. Ranking via robust binary classification. In *Advances in Neural Information Processing Systems*, pages 2582–2590, 2014.