How to Intelligently Distribute Training Data to Multiple Compute Nodes: Distributed Machine Learning via Submodular Partitioning

Kai Wei[1] **Rishabh Iyer**[1] **Shengjie Wang**[2] **Wenruo Bai**[1] **Jeff Bilmes**[1]

[1] Department of Electrical Engineering, University of Washington
[2] Department of Computer Science, University of Washington
{kaiwei, rkiyer, wangsj, wrbai, bilmes}@u.washington.edu

Abstract

In this paper we investigate the problem of training data partitioning for parallel learning of statistical models. Motivated by [10], we utilize submodular functions to model the utility of data subsets for training machine learning classifiers and formulate this problem mathematically as submodular partitioning. We introduce a simple and scalable greedy algorithm that near-optimally solves the submodular partitioning problem. We empirically demonstrate the efficacy of the proposed algorithm to obtain data partitioning for distributed optimization of convex and deep neural network objectives. Empirical evidences suggest that the intelligent data partitioning produced by the proposed framework leads to faster convergence in the case of distributed convex optimization, and better resulting models in the case of parallel neural network training.

1 Introduction

Big data presents significant computational challenges to machine learning since, while big data is still getting bigger, it is expected that we are nearing the end of Moore's law [8], and single threaded computing speed has unfortunately not significantly improved since about 2003. It is hence imperative to develop efficient and scalable methods for large scale training of statistical models. Parallel and distributed computing approaches are natural for this challenge.

Since machine learning procedures are performed over sets of data, one simple way to achieve parallelism is to split the data into chunks each of which resides on a compute node. This is the idea behind many parallel learning approaches such as ADMM [1] and distributed neural network training [5], to name only a few. Such parallel schemes are often performed where the data samples are distributed to their compute nodes in an arbitrary or random fashion. However, there has apparently been very little work on how to intelligently split the data to ensure that the resultant model can be learned in a more efficient manner.

One way to approach this problem is to consider a class of "utility" functions on the training data. Given a set $V = \{v_1, \dots, v_n\}$ of training data items, suppose that we have a set function $f : 2^V \to \mathbb{R}_+$ that measures the utility in subsets of the data set V. That is, given any $A \subseteq V$, $f(A)$ measures the utility of the training data subset A for producing a good resulting trained model. Given a parallel training scheme (e.g., ADMM) with m compute nodes, we consider an m-partition $\pi = (A_1^\pi, A_2^\pi, \dots, A_m^\pi)$ (i.e., $\cup_i A_i^\pi = V$ and $\forall i \neq j, A_i^\pi \cap A_j^\pi = \emptyset$) of the entire training data V, where we send the i^{th} block A_i^π of the partition π to the i^{th} compute node. If each compute node i has a block of the data A_i^π that is non-representative of the utility of the whole (i.e., $f(A_i^m) \ll f(V)$), the parallel learning algorithm, at each iteration, might have the compute nodes deduce models that are widely different from each other. Any subsequent aggregation of the models could then result in a joint model that is non-representative of the whole, especially in a non-convex case like deep neural network models. On the other hand, suppose that an intelligent partition π of the training data V is achieved such that each block A_i^π is highly representative of the whole (i.e., $f(A_i^m) \approx f(V), \forall i$). In this case, the models deduced at the compute nodes will tend to be close to a model trained on the entire data, and any aggregation of the resulting models (say via ADMM) will likely be better. An intelligent data partition, in fact, may have a positive effect in two ways: 1) it can lead to a better final solution (in the non-convex case), and 2) faster convergence may be achieved even in the convex case thanks to the fact that any "oscillations" between a distributed stage (where the compute nodes are operating on local data) and an aggregation stage (where some form of model average is computed) could be dampened and hence reduced.

In this work, based on the above intuition, we mathematically describe this goal as obtaining an m-partition of the data set V such that the worst-case or the average-case utility among all blocks in the partition is maximized. More precisely, this can be formulated as follows:

$$\max_{\pi \in \Pi} \Big[(1-\lambda) \min_i f_i(A_i^\pi) + \frac{\lambda}{m} \sum_{j=1}^{m} f_j(A_j^\pi) \Big], \tag{1}$$

where the set of sets $\pi = (A_1^\pi, A_2^\pi, \cdots, A_m^\pi)$ forms a partition of the finite set V, Π refers to the set of all partitions of V into m blocks, and f_i models the utility score assigned to i^{th} block A_i^π of the partition. The first term $\min_i f_i(A_i^\pi)$ valuates the worst-case utility among the blocks in the partition π, while the second term $\frac{1}{m}\sum_{i=1}^m f_i(A_i^\pi)$ measures the averaged utility of the partition. The parameter λ controls the objective: $\lambda = 1$ is the average case, $\lambda = 0$ is the robust case, and $0 < \lambda < 1$ is a mixed case. We further categorize Problem 1 on if the f_i's are identical to each other (*homogeneous*) or not (*heterogeneous*). In general, Problem 1 is hopelessly intractable, even to approximate, but we assume that the f is monotone non-decreasing (i.e., $f(S) \leq f(T)$ whenever $S \subseteq T$), normalized ($f(\emptyset) = 0$), and submodular [2] (i.e., $\forall S, T \subseteq V$, $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$). In this case, we refer to Problem 1 as a form of *Submodular Partitioning*. These assumptions allow us to develop fast, simple, and scalable algorithms that have approximation guarantees, as is fully developed in [11]. These assumptions, moreover, allow us to retain the naturalness and applicability of Problem 1 to the goal at hand. Submodular functions are an ideal class of functions for modeling information over data sets. For example, [10] show that the utility functions of data subsets for training certain machine learning classifiers can be derived as submodular functions. If f is selected as the class of submodular functions that appropriately model the notion of utility for a given machine learning setting (which could be different depending, say, on what form of training one is doing), solving the homogeneous instance of Problem 1 with f as the objective, then, addresses a core goal in modern machine leanring on big data sets, namely how to intelligently partition and distribute training data to multiple compute nodes.

It should be noted that a random partition might have a high probability of performing well, and might have exponentially small probability of producing a partition that is worst case. On the other hand, there are also quite likely a small number of partitions that perform exceedingly well and a random partition also has exponentially small probability of landing on one of these high quality partitions. Our quest is to develop methods that increase the likelihood that we can discover one of these rare but high performing partitions. Moreover, there are other applications (i.e., locality maximization via bipartite graphs, where the goal is to organize the data so as to improve data locality, as in the work of Smola and also the NIPS2015 work of Iglesias) where a random partition is quite likely to perform very poorly. This again is an application of our work.

Our main paper [11] fully describes all algorithms, theorems demonstrating mathematical guarantees, and related work regarding both Problem 1 and also its dual companion problem that is useful for other applications. In this workshop paper, we concentrate on the aforementioned application. We omit the discussion of the previous work on Problem 1, which has been described in [11]. In general, much of the previous work focuses on designing theoretically tighter algorithms that are not as scalable. Moreover, in the below we concentrate on the robust $\lambda = 0$ case under the homogeneous setting (f_i's are identical to each other) for now, although we believe that more experiments with the right submodular functions will show some utility in the $0 < \lambda < 1$ case as well.

The paper is organized as follows: We first introduce a practical and scalable greedy algorithm for solving Problem 1 with $\lambda = 0$ under the homogeneous setting. We then empirically demonstrate the efficacy of the proposed algorithm for obtaining data partitions in parallel machine learning schemes, such as ADMM and distributed neural network training.

2 An Approximation Algorithm for Problem 1 with $\lambda = 0$

Notation: we define $f(j|S) \triangleq f(S \cup j) - f(S)$ as the gain of $j \in V$ in the context of $S \subseteq V$. We assume w.l.o.g. that the ground set is $V = \{v_1, v_2, \cdots, v_n\}$. The homogeneous instance of Problem 1 with $\lambda = 0$ can be equivalently stated as follows:

$$\max_{\pi \in \Pi} \min_i f(A_i^\pi), \tag{2}$$

GREEDMAX: We approach this problem from the perspective of the greedy algorithm. In particular we introduce GREEDMAX as described in Alg 1. The key idea of GREEDMAX is to greedily add an item with the maximum marginal gain to the block whose current solution is minimum. Initializing $\{A_i\}_{i=1}^m$ with the empty sets, the greedy flavor also comes from that it incrementally grows the solution by greedily improving the overall objective $\min_{i=1,\dots,m} f_i(A_i)$ until $\{A_i\}_{i=1}^m$ forms a partition. Besides its simplicity, Theorem 2.1 offers the optimality guarantee.

Theorem 2.1. *Under the homogeneous setting ($f_i = f$ for all i),* GREEDMAX *is guaranteed to find a partition $\hat{\pi}$ such that* $\min_{i=1,\dots,m} f(A_i^{\hat{\pi}}) \geq \frac{1}{m} \max_{\pi \in \Pi} \min_{i=1,\dots,m} f(A_i^\pi)$.

Algorithm 1: GREEDMAX

```
1: Input: $f, m, V$.
2: Let $A_1 =, \dots, = A_m = \emptyset$; $R = V$.
3: while $R \neq \emptyset$ do
4:     $j^* \in \text{argmin}_j f(A_j)$;
5:     $a^* \in \text{argmax}_{a \in R} f(a|A_{j^*})$
6:     $A_{j^*} \leftarrow A_{j^*} \cup \{a^*\}$;
7:     $R \leftarrow R \setminus a^*$
8: end while
9: Output $\{A_i\}_{i=1}^m$.
```

All proofs for this paper are given in [11]. By assuming the homogeneity of the f_i's, we obtain a very simple $1/m$-approximation algorithm improving upon the state-of-the-art factor $1/(2m-1)$ on this problem as shown in [3]. Moreover, thanks to the lazy evaluation trick as described in [4], Line 5 in Alg. 1 need not to recompute the marginal gain for every item in each round, leading GREEDMAX to scale to large data sets.

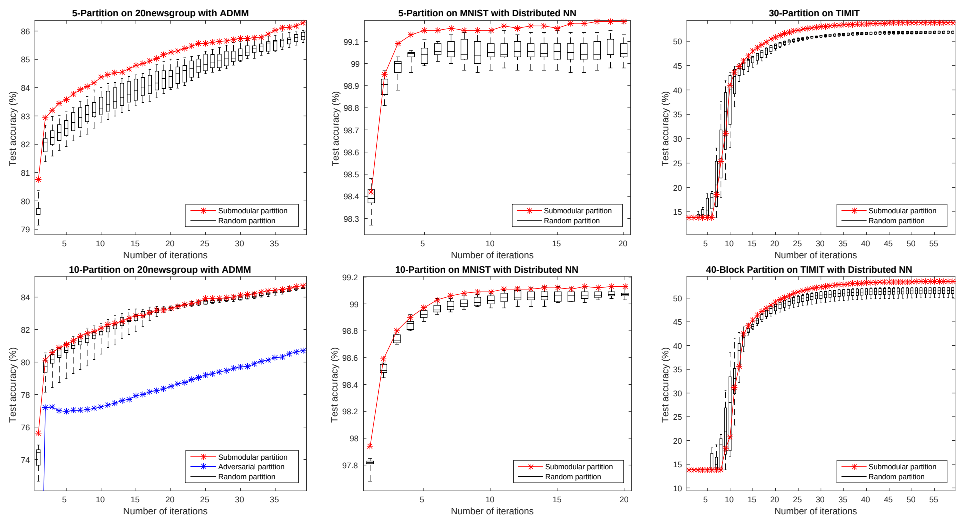
In the context of parallel machine learning, obtaining a partition for Problem 1 using GREEDMAX can be viewed as an initial pass on the entire training data to distribute subsets of it to compute nodes before the parallel computation begins. Although the proposed algorithm GREEDMAX is scalable even to very large data sets, it should be noted that there are still computational costs associated with this initialization procedure. However, many statistical learning methods (e.g., the EM algorithm for training Gaussian Mixtures Models, or the back-propagation algorithm for training deep neural networks) typically involve many training passes over the entire data set. Moreover, during model development (when a researcher is uncertain about the optimal model topology), a number of model topologies need to be tested until the best setting is discovered, even for tools such as [6]. We contend that any cost associated with the initial pass over the data (solving Problem 1 for an intelligent partition) can be amortized over the life of the utility of the resulting partition since a smart partition will speed up training time and may even lead to a better overall model in the non-convex case.

3 Experiments

In this section we empirically evaluate the proposed algorithm GREEDMAX on real-world data partitioning applications including distributed ADMM and distributed deep neural network training.

3.1 Distributed Convex Optimization

We first consider data partitioning for distributed convex optimization. We evaluate on a text categorization task. We use 20 Newsgroup data set, which consists of 18,774 articles divided almost evenly across 20 classes. The text categorization task is to classify an article into one newsgroup (of twenty) to which it was posted. We randomly split 2/3 and 1/3 of the whole data as the training and test data. The task is solved as a multi-class classification problem, which we formulate as an L-2



(a)20Newsgroups (b) MNIST (c) TIMIT

Figure 1: Comparison between submodular and random partitions for distributed ML, including ADMM (Fig 1a) and distributed deep neural nets (Fig 1b) and (Fig 1c). For the box plots, the central mark is the median, the box edges are 25th and 75th percentiles, and the bars indicate the best and worst cases. In the right two columns, the adversarial partitions are so bad that they are off the plots.

regularized logistic regression. We solve this convex optimization problem in a distributive fashion, where the data samples are partitioned and distributed across multiple machines. In particular we implement an ADMM algorithm as described in [1] to solve the distributed convex optimization problem.

We formulate the data partitioning problem as an robust instance ($\lambda = 0$) of Problem 1 under the homogeneous setting. In the experiment, we solve the data partitioning using GreedMax. We model the utility of a data subset using the feature-based submodular function [12, 10, 9], which has the form:

$$f_{\text{fea}}(A) = \sum_{u \in \mathcal{U}} m_u(V) \log m_u(A), \tag{3}$$

where $\mathcal{U}$ is the set of "features", $m_u(A) = \sum_{a \in A} m_u(a)$ with $m_u(a)$ measuring the degree that the article a possesses the feature $u \in \mathcal{U}$. In the experiments, we define $\mathcal{U}$ as the set of all words occurred in the entire data set and $m_u(a)$ as the number of occurrences of the word $u \in \mathcal{U}$ in the article a. f_{fea} is in the form of a sum of concave over modular functions, hence is monotone submodular [7]. The class of feature-based submodular function has been widely applied to model the utility of a data subset on a number of tasks, including speech data subset selection [12, 13], and image summarization [9]. Moreover f_{fea} has been shown in [10] to model the log-likelihood of a data subset for a Naïve Bayes classifier.

We compare the submodular partitioning with the random partitioning for $m = 5$ and $m = 10$. We test with 10 instances of random partitioning. The results are plotted in Fig 1a. For $m = 10$ we also run an instance on an adversarial partitioning, where each block is formed by grouping every two of the 20 classes in the training data. We observe submodular partitioning converges faster than the random partitioning, both of which perform significantly better than the adversarial partition. In particular significant and consistent improvement over the best of 10 random instances is achieved by the submodular partition across all iterations when $m = 5$.

3.2 Distributed Deep Neural Networks (DNN)

Next we evaluate our framework on the distributed deep neural network (DNN) training. We test on two tasks: 1) handwritten digit recognition on the MNIST database; 2) phone classification on the TIMIT data. The data for the handwritten digit recognition task consists of 60,000 training and 10,000 test samples. Each data sample is an image of handwritten digit. The training and test data are almost evenly divided into 10 different classes. For the phone classification task, the data consists of 1,124,823 training and 112,487 test samples. Each sample is a frame of speech. The training data is divided into 50 classes, each of which corresponds to a phoneme. The goal of this task to classify each speech sample into one of the 50 phone classes.

A 4-layer DNN model is applied for the MNIST experiments, and we train a 5-layered NN for the TIMIT experiments. We apply the same distributed training procedure for both tasks. Given a partitioning of the training data, we distributively solve m instances of sub-problems in each iteration. We define each sub-problem on a separate block of the data. We employ the stochastic gradient descent as the solver on each instance of the sub-problem. In the first iteration we use a randomly generated model as the initial model shared among the m sub-problems. Each sub-problem is solved with 10 epochs of the stochastic gradient decent training. We then average the weights in the m resultant models to obtain a consensus model, which is used as the initial model for each sub-problem in the successive iteration. Note that this distributed training scheme is similar to the one presented in [5].

The submodular partitioning for both tasks is obtained by solving the homogeneous case of Problem 1 with $\lambda = 0$ using GREEDMAX on a form of clustered facility location $f_{\text{c-fac}}$, as proposed and used in [10]. The function is defined as follows:

$$f_{\text{c-fac}}(A) = \sum_{y \in \mathcal{Y}} \sum_{v \in V^y} \max_{a \in A \cap V^y} s_{v,a}, \tag{4}$$

where $s_{v,a}$ is the similarity measure between sample v and a, $\mathcal{Y}$ is the set of class labels, and V^y is the set of samples in V with label $y \in \mathcal{Y}$. Note $\{V^y\}_{y \in \mathcal{Y}}$ forms a disjoint partitioning of the ground set V. In both the MNIST and TIMIT experiments we compute the similarity $s_{v,a}$ as the RBF kernel between the feature representation of v and a. [10] show that $f_{\text{c-fac}}$ models the log-likelihood of a data subset for a Nearest Neighbor classifier. They also empirically demonstrate the efficacy of $f_{\text{c-fac}}$ in the case of neural network based classifiers.

We run 10 instances of random partitioning as the baseline. As shown in Figure 1b and 1c, the submodular partitioning significantly outperforms the random baseline. For all cases, we observe that better resulting models are obtained by using the submodular partitioning than all 10 instances of the random partitioning. On the other hand, the adversarial partitioning, which is formed by grouping items with the same class, cannot even be trained in both cases.

4 Conclusions

In this paper we present a framework for splitting training data intelligently as an initial step to existing parallel statistical learning paradigms. The framework is formulated as a submodular partitioning problem, where we utilize appropriate submodular functions to model the utility of data subsets for training machine learning classifiers. We propose a simple and efficient greedy algorithm for solving the submodular partitioning problem. We give empirical validation of the proposed approach on both distributed convex optimization and parallel neural network training across a number of machine learning tasks, including text categorization, handwritten digit recognition, and phone classification. Consistent and significant improvements over the principle of random partitioning are achieved by the proposed intelligent data distribution framework.

References

[1] 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*, 2011.

[2] S. Fujishige. *Submodular functions and optimization*, volume 58. Elsevier, 2005.

[3] S. Khot and A. Ponnuswami. Approximation algorithms for the max-min allocation problem. In *APPROX*, 2007.

[4] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, 1978.

[5] D. Povey, X. Zhang, and S. Khudanpur. Parallel training of deep neural networks with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*, 2014.

[6] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[7] P. Stobbe and A. Krause. Efficient minimization of decomposable submodular functions. In *NIPS*, 2010.

[8] S. E. Thompson and S. Parthasarathy. Moore's law: the future of si microelectronics. *materials today*, 9(6):20–25, 2006.

[9] S. Tschiatschek, R. K. Iyer, H. Wei, and J. A. Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems*, pages 1413–1421, 2014.

[10] K. Wei, R. Iyer, and J. Bilmes. Submodularity in data subset selection and active learning. In *ICML*, 2015.

[11] K. Wei, R. Iyer, S. Wang, W. Bai, and J. Bilmes. Mixed robust/average submodular partitioning: Fast algorithms, guarantees, and applications: NIPS 2015 Extended Supplementary.

[12] K. Wei, Y. Liu, K. Kirchhoff, C. Bartels, and J. Bilmes. Submodular subset selection for large-scale speech training data. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Florence, Italy, 2014.

[13] K. Wei, Y. Liu, K. Kirchhoff, and J. Bilmes. Unsupervised submodular subset selection for speech data. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Florence, Italy, 2014.