Distributed training of Large-scale Logistic models

Siddharth Gopal SGOPAL1@CS.CMU.EDU
5000 Forbes Ave, Carnegie Mellon University, Pittsburgh PA 15217 USA

Yiming Yang YIMING@CS.CMU.EDU
5000 Forbes Ave, Carnegie Mellon University, Pittsburgh PA 15217 USA

Abstract

Regularized Multinomial Logistic regression has emerged as one of the most common methods for performing data classification and analysis. With the advent of large-scale data it is common to find scenarios where the number of possible multinomial outcomes is large (in the order of thousands to tens of thousands) and the dimensionality is high. In such cases, the computational cost of training logistic models or even simply iterating through all the model parameters is prohibitively expensive. In this paper, we propose a training method for large-scale multinomial logistic models that breaks this bottleneck by enabling parallel optimization of the likelihood objective. Our experiments on large-scale datasets showed an order of magnitude reduction in training time.

1. Introduction

Regularized Multinomial Logistic regression (RMLR) is one of the fundamental tools to model the the probabilities of occurrence among one or more discrete outcomes or class-labels. It is the choice algorithm for classification in several fields such as economics to model customer retention, purchase propensity etc (Rust & Zahorik, 1993), in biomedical domains to model probability of a disease occurrence (Kirkwood et al., 1988), various engineering areas to model machine failure probability etc.

With the advent of large-scale data, there is growing interest in enabling such fundamental models such as RMLR to cope with large-scale factors such as large number of class-labels, high dimensionalities of the input space and large-number of training instances. Large-scale data is not uncommon; for example, the Image-net database which is a repository of images from the web has about 14 million images (instances) organized into one of 20,000 words (class-labels) from the word-net hierarchy. Another example is the Tiny-image collection which is a collection of 80 million low-scale 32x32 images from the web organized into a 75,000 abstract noun classes in English. In the text domain, the Open Directory Project (ODP) has existed for a long time and has amassed 4.6 million web-pages into a human-browsable hierarchy of 1 million classes. Wikipedia, where it is easy to get reasonably high quality labels for free, has millions of pages labeled with multiple classes. For example, in the some of the recently released data collections[1] based on wikipedia and ODP, the number of classes range from tens of thousands to hundreds of thousands.

In such large-scale scenarios, it could be prohibitively expensive to train RMLR models. To quantify the scale of data we are interested in, let us consider a recently released web-scale data collection[1]. The data is a subset of webpages from the ODP [2]. There are about 100,000 webpages organized into one of 12,000 classes and each webpage is represented by a sparse vector of 350,00 word-level features. To train a RMLR model, one would need to learn $12000 \times 350000 = 4.2$ billion parameters which close to 17GB of parameters. Even if we overcome the hurdle of storing such large number of parameters in memory, it would still take significant amount of time to even perform a single iteration over all parameters. It is practically infeasible to learn a RMLR model with such large number of parameter on a single computer. The problem could be further amplified if we have a large number of training instances. However, the primary focus of our paper will

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

[1] http://lshtc.iit.demokritos.gr/
[2] http://dmoz.org

be on how to scale when the number of classes is large. More specifically, we will address the issue of how to cope up with the memory and computational requirements of training RMLR model under such large-scale scenarios.

In this paper, we explore a distributed approach to train RMLR models. The main chunk of the computation is accomplished by optimizing sets of parameters parallely across different computing units thereby gaining two big advantages (a) Faster computation - we can use the resources from multiple computing units to solve different parts of the single optimization problem (b) Distributed Memory - parameters can now be spread across multiple computing units simultaneously. The key idea is to replace the log partition function of the multinomial logit with a parallelizable upper-bound based on the concavity of the log-function with several variational parameters. The procedure is an iterative procedure where in one step, the variational parameters are optimized to give the best possible upper-bound, and in the other step, the different sets of parameters are optimized parallely. Furthermore, we prove that this iterative procedure to the optimize the upper bounded objective converges to the same optimal solution as the original RMLR model. To our knowledge, this is the first work that shows that RMLR models can be scaled to datasets with tens of thousands of class-labels and billions of parameters in a matter of few hours.

2. Related work

To our knowledge, there is no directly related work that addresses the problem of training RMLR models for large number of classes. However, we outline some of the most commonly used methods to train RMLR models and discuss the limitation of such models in large-scale settings.

The simplest approach to fit RMLR models are first order methods such as gradient descent. However, the problem with gradient methods is that they require careful tuning of the step-size and are generally slow to converge. Second order methods such as Newton's method improve over gradient methods in the sense that the descent direction provides a natural unit step-length and provide exponential convergence near the optimal solution. An efficient implementation of newton's method known as iterated reweighted least squares (Holland & Welsch, 1977) is widely used to fit logistic models. However second order methods fail to be useful in high dimensional settings as the calculation of descent direction involves a high dimensional matrix inversion (inverse of the Hessian) which is generally computationally intractable. Quasi newton methods such as BFGS (Shanno, 1985) and its limited memory variant LBFGS (Liu & Nocedal, 1989) are second order methods which overcome this problem of matrix inversion by continually updating an approximate inverse of the Hessian - although at a slower convergence rate. Recently, multiple studies (Sha & Pereira, 2003), (Schraudolph et al., 2007), (Daumé III, 2004) have shown that LBFGS methods offer the best trade-off between convergence and computational requirements and authors have often adopted it as the standard choice to train logistic models. However, in the large-scale setting we have at hand, even LBFGS does not meet the computational challenges. Firstly, LBFGS is not an inherently parallelizable (across parameters) algorithm. Secondly, due to the requirement of repeated line-searches and function value evaluation, the parameters to be learnt typically have to be stored in memory (reading and writing 17GB of parameters to disk is expensive). Thirdly, LBFGS requires storing the gradient values from the last few iterations which further increase the memory requirements; for example if the gradient from the last 5 iterations is stored, the memory requirement shoots upto $17 \times 5 \sim 85$GB making it practically infeasible.

Other ways to train RMLR are using iterative scaling (Darroch & Ratcliff, 1972), conjugate gradient (Lin et al., 2008) and dual methods (Yu et al., 2011). To our knowledge, there has been no work on large-scale distributed RMLR training using these methods particularly to the scale we are interested in. Also refer (Minka, 2003) for an excellent comparison between the different existing approaches for *binary* logistic regression.

Indirectly related to this paper are a few works in stochastic gradient descent (Zhang, 2004), (Bottou, 2010) and online learning (Bottou, 1998) . Stochastic and online approaches address scalability issues when the number of training instances is large. This is very different from our goal where the focus is large number of classes rather than large number of training instances. Moreover, unlike stochastic approaches our aim is not to provide guarantees on expected (expectation w.r.t data) error of the model but training a RMLR model on a fixed training dataset.

3. Training Large-scale RMLR

Let $\{x_i, t_i\}_{i=1}^{i=N}$ denote the set of training instances where each $x_i \in \mathcal{R}^D$ and $t_i \in \{1, 2, ..K\}$ and K denotes the number of class labels. Define the indicator $y_{ik} = I(t_i = k)$ which denotes whether the i^{th} training example belongs to class k or not. Let the probability

of a given instance x to have a class-label k be modelled as,

$$P(y=k|x) = \frac{exp(w_k^\top x)}{\sum_{k'=1}^{K} exp(w_{k'}^\top x)}$$

where $\mathbf{W} = \{w_1, w_2, ..w_K\}$ denote the set of parameters. The training objective of RMLR can be written as,

$$OPT1$$
$$\min_{\mathbf{W}} \; G_{\mathcal{D}}(W)$$
$$G_{\mathcal{D}}(W) = \frac{\lambda}{2}\sum_{k=1}^{K}\|w_k\|^2 - \sum_{k=1}^{K}\sum_{i=1}^{N} y_{ik} w_k^\top x_i + \sum_{i=1}^{N}\log(\sum_{k'=1}^{K}\exp(w_{k'}^\top x_i))$$

The most natural way to parallelize is to optimize each class-level parameter w_k in parallel. However this is not directly possible due to the presence of the log partition function which couples all the class-level parameters together inside a log-sum-exp function. This makes the objective non decomposable across the w_k's. This leads to the question - can we replace the log-partition function by a *parallelizable* function ? Secondly, can this *parallelizable* function also be an upper-bound to the log-partition function as this would guarantee that true-minimum is at most as high as the minimum of the upper bounded objective. And finally, the introduction of this parallelizable function must not make the problem *harder* to solve (like for example make it non-differentiable or multiple local minima etc).

To this end, we explore 3 different bounds for the log partition function and their applicability to the problem at hand.

3.1. Piece-wise Bounds

One of the properties of convex functions is that they can be approximated by piece-wise linear functions to any degree of precision just by increasing the number of pieces. This property can be used to upper and lower-bound the log-sum-exp function (Hsiung et al., 2008). The idea is to find a set of variational parameters a, b, c, d such that

$$\max_j\{a_j^\top\gamma + b_j\} \leq \log\left(\sum_{k=1}^{K}\exp(\gamma_k)\right) \leq \max_{j'}\{c_{j'}^\top\gamma + d_{j'}\}$$
$$a, c \in \mathcal{R}^K \quad b, d \in \mathcal{R}$$

A similar idea was also used in (Marlin et al., 2011) to approximate a binary logistic function but with a quadratic function instead of linear functions. However there a few problems associated with such piece-wise bounds. Firstly, finding the variational parameters a, b, c, d is not easy. Till date a constructive solution for arbitrary precision for the log-sum-exp function has been established only for $K = 2$ (Hsiung et al., 2008). Without a constructive solution one needs to resort to derivative free optimization methods like Nelder Mead method etc to fix the value of the variational parameters. More over the number of such variational parameters also grows linearly with the number of classes K, therefore for datasets with a large number of classes, finding the variational parameters through derivative free methods would be even harder. Secondly, even if one were to find such parameters, the objective function is still not parallelizable across the class-level parameters. Thirdly, using this bound would introduce non-differentiability in the objective i.e. the log-sum-exp function would be replaced by a max over linear functions, thereby rendering the objective non-differentiable. Due to the above issues, piece-wise linear bounds do not help us achieve distributed training of large-scale RMLR models.

3.2. Log-concavity bound

This is a well known bound that exploits the first order concavity property of the log-function. It has been used in many works including (Blei & Lafferty, 2006), (Bouchard, 2007). The bound is as follows,

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

where a is a variational parameter. Minimizing the RHS over a gives $a = \frac{1}{\gamma}$ and makes the bound tight. To incorporate this into the objective, we introduce variational parameters a_i for each training instance i. We have the log-partition function for instance i bounded as,

$$\log\left(\sum_{k=1}^{K}\exp(w_k^\top x_i)\right) \leq a_i\sum_{k=1}^{K}\exp(w_k^\top x_i) - log(a_i) - 1$$

Firstly, note that incorporating this into the objective makes the objective parallelizable across w_k's. That is, for a given fixed value of the variational parameter a_i, the optimization over $\mathbf{W}$ splits into a sum of K different objectives. Specifically, the class-level parameter for class k i.e. w_k can be optimized as

$$\arg\min_{w_k} \quad \frac{\lambda}{2}\|w_k\|^2 - \sum_{i=1}^{N} y_{ik} w_k^\top x_i + \sum_{i=1}^{N} a_i \exp(w_k^\top x_i) \tag{2}$$

Secondly, note that finding the variational parameter is an easy problem, because optimizing over a_i in (1)

has a closed form solution. Thirdly note that the combined objective (as given below), is still differentiable.

$$
\begin{aligned}
&OPT\ 2\\
&\min_{\mathbf{A}>0,\mathbf{W}} \quad F_{\mathcal{D}}(W,A)\\
&F_{\mathcal{D}}(W,A) \quad = \tfrac{\lambda}{2}\sum_{k=1}^{K}\|w_k\|^2 + \sum_{i=1}^{N}\Big[-\sum_{k=1}^{K} y_{ik} w_k^\top x_i\\
&\qquad\qquad +a_i\sum_{k=1}^{K}\exp(w_k^\top x_i) - \log(a_i) - 1\Big]
\end{aligned}
$$

The only down-side of this bound is that the convexity of the original objective is now lost due to the presence of a product of linear and exponential function i.e. $a_i \exp(w_k^\top x_i)$. This introduces the possibility of potentially multiple local minima in the objective and hence the upper-bound could be potentially loose.

However, we show that it is still possible to reach the optimal solution of the OPT 1 through OPT 2 by showing the following,

1. There is exactly one stationary point of the combined objective $F_{\mathcal{D}}(W,A)$.
2. This single stationary point of $F_{\mathcal{D}}(W,A)$ is a feasible solution to OPT 2 and coincides with the optimal solution of OPT 1.
3. A block co-ordinate descent procedure that converges to the stationary point of $F_{\mathcal{D}}(W,A)$ i.e optimal solution of OPT 1.

Let us consider one of the (possibly many) stationary points of $F_{\mathcal{D}}(W,A)$: $\mathbf{W}^*, \mathbf{A}^*$. Stationarity implies that the gradient is zero at this point,

$$
\begin{aligned}
&\frac{\partial F_{\mathcal{D}}}{\partial a_i}|_{\mathbf{W}^*,\mathbf{A}^*} = 0 \Rightarrow\\
&A.\quad a_i^* = \frac{1}{\sum_{k=1}^{K}\exp(w_k^{*\top} x_i)} \qquad (3)\\
&B.\ \lambda\sum_{k=1}^{K} w_k^* - \sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}x_i + \sum_{i=1}^{N}\frac{\exp(w_{t_i}^{*\top} x_i)}{\sum_{k'=1}^{K}\exp(w_{k'}^{*\top} x_i)} x_i = 0 \qquad (4)
\end{aligned}
$$

From equation (3), it follows that the stationary point $\mathbf{A}^*$ is a feasible solution to the optimization problem OPT 2. Secondly, from equation (4) it can be seen that the stationary point $\mathbf{W}^*$ also satisfies the first order stationarity conditions of $G_{\mathcal{D}}(W)$. But $G_{\mathcal{D}}(W)$ is strictly convex objective and therefore has exactly one unique stationary point which is a minimum and is the optimal solution of $OPT1$. Therefore, all the stationary points of $F_{\mathcal{D}}(W,A)$ have the same $\mathbf{W}^*$. And given that $\mathbf{A}$ is also uniquely defined by $\mathbf{W}$, there is exactly one stationary point of $F_{\mathcal{D}}(W,A)$, which coincides with optimal solution to OPT 2.

Algorithm 1 Block coordinate descent algorithm

Initialize : $t \leftarrow 0, \mathbf{A}_0 \leftarrow \frac{1}{K}, \mathbf{W}_0 \leftarrow 0$.
Result : $\mathbf{W}^t$

While : Not converged
In parallel : $\mathbf{W}^{t+1} \leftarrow \arg\min_W F_{\mathcal{D}}(W, \mathbf{A}^t)$
$\mathbf{A}^{t+1} \leftarrow \arg\min_A F_{\mathcal{D}}(\mathbf{W}^{t+1}, A)$
$t \leftarrow t+1$

So far we have established that although $F_{\mathcal{D}}(\mathbf{W},\mathbf{A})$ is non-convex, it has a single unique stationary point. Next we outline a block co-ordinate descent algorithm that iteratively optimizes $\mathbf{W}$ and $\mathbf{A}$ and show that this block co-ordinate descent converges to a stationary point of $F_{\mathcal{D}}(W,A)$.

3.2.1. Convergence

Let us define the following functions,

$$
\begin{aligned}
F_{\mathbf{W}^t}(A) &= F_{\mathcal{D}}(\mathbf{W}^t, A)\\
F_{\mathbf{A}^t}(W) &= F_{\mathcal{D}}(W, \mathbf{A}^t)
\end{aligned}
$$

Now, it can be seen that the sequence of iterates $\{\mathbf{W}_t, \mathbf{A}_t\}$ are generated as follows,

$$
\begin{aligned}
&\mathbf{W}^0 = 0, \quad \mathbf{A}^0 = \frac{1}{K}\\
&\mathbf{W}^{t+1} = \arg\min_W F_{\mathbf{A}^t}(W)\\
&\mathbf{A}^{t+1} = \arg\min_A F_{\mathbf{W}^{t+1}}(A) \ \textbf{i.e.}\ a_i^{t+1} = \frac{1}{\sum_{k=1}^{K}\exp(w_k^{t+1\top} x_i)}
\end{aligned} \qquad (5)
$$

Note that $F_{\mathbf{A}^t}(W)$ is strongly convex with parameter λ. Therefore by strong convexity we have,

$$
F_{\mathbf{A}^t}(\mathbf{W}^t) - F_{\mathbf{A}^t}(\mathbf{W}^{t+1}) \geq \frac{\lambda}{2}||\mathbf{W}^t - \mathbf{W}^{t+1}||^2 \qquad (6)
$$

Also, since $\mathbf{A}^{t+1}$ minimizes, $F_{\mathbf{W}^{t+1}}(\mathbf{A})$, we have,

$$
F_{\mathbf{W}^{t+1}}(\mathbf{A}^t) - F_{\mathbf{W}^{t+1}}(\mathbf{A}^{t+1}) \geq 0 \qquad (7)
$$

Adding (6) and (7), and using the fact that $F_{\mathbf{A}^t}(\mathbf{W}^{t+1}) = F_{\mathbf{W}^{t+1}}(\mathbf{A}^t)$, and summing over T iterations,

$$
F_{\mathcal{D}}(\mathbf{W}^0,\mathbf{A}^0) - F_{\mathcal{D}}(\mathbf{W}^{t+1},\mathbf{A}^{t+1}) \geq \frac{\lambda}{2}\sum_{t=1}^{T}||\mathbf{W}^t - \mathbf{W}^{t+1}||^2
$$

Since $F_{\mathcal{D}}$ is bounded from below i.e $F_{\mathcal{D}} > 0$, we have the the L.H.S is bounded. Therefore,

$$
\lim_{t\to\infty}\tfrac{\lambda}{2}\sum_{t=1}^{T}||\mathbf{W}^t - \mathbf{W}^{t+1}||^2 < \infty \Rightarrow \mathbf{W}^t \to \mathbf{W}'
$$

Since (5) is a continuous function, it preserves limits, therefore $\mathbf{A}^t \rightarrow \mathbf{A}^{'}$. Hence the sequence of iterates generated by the block coordinate descent algorithm converges.

3.2.2. Convergence to Optimal solution

Let us consider the gradient of $F_{\mathbf{A}^{t+1}}(W)$ at $\mathbf{W}^{t+1}$

$$\frac{\partial F_{\mathbf{A}^{t+1}}(W)}{\partial W} = \lambda \sum_{k=1}^{K} w_k - \sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} x_i + \sum_{i=1}^{N} a_i^{t+1} \exp({w_k}^\top x_i) x_i$$

Using the fact that $\frac{\partial F_{\mathbf{A}^t}}{\partial W}\Big|_{\mathbf{W}^{t+1}} = 0$, we have that

$$\left\| \frac{\partial F_{\mathbf{A}^{t+1}}(.)}{\partial W}|_{\mathbf{W}^{t+1}} \right\| = \left\| \sum_{i=1}^{N} (a_i^{t+1} - a_i^t) \exp({w_k^{t+1}}^\top x_i) x_i \right\|$$

$$\leq \sum_{i=1}^{N} \|a_i^t - a_i^{t+1}\| \exp({w_k^{t+1}}^\top x_i) \|x_i\| \tag{8}$$

Since $exp(x)$ is a continuous function preserving limits, and $\mathbf{W}$,$\mathbf{A} \rightarrow \mathbf{W}^{'}, \mathbf{A}^{'}$ the RHS of the above goes to zero. Also, from the definition of $\mathbf{A}^{t+1}$

$$\left\| \frac{\partial F_{\mathcal{D}}(\mathbf{W}^{t+1}, A)}{\partial A}\Big|_{\mathbf{A}^{t+1}} \right\| = 0 \tag{9}$$

Combining eq (8), eq (9) and taking limits we have that,

$$\left\| \frac{\partial F_{\mathcal{D}}(W, A)}{\partial W, A}\Big|_{\mathbf{W}^{t+1}, \mathbf{A}^{t+1}} \right\| \rightarrow 0 \quad \text{as } t \rightarrow \infty$$

From above, we have that as $t \rightarrow \infty$, the partial gradients tend to zero therefore the sequence converges to the stationary point of $F_{\mathcal{D}}(W, A)$ which is the optimal solution of OPT 1, that is, $\mathbf{W}^{'} = \mathbf{W}^{*}$.

3.3. Double majorization bound

This bound was proposed in (Bouchard, 2007) to enable variational inference for logit models with Gaussian priors. The bound is given by,

$$\log\left(\sum_{i=1}^{K} \exp(w_k^\top x_i)\right) \leq a_i + \sum_{k=1}^{K} \log(1 + e^{w_k^\top x_i - a_i})$$

where the variational parameters $a_i \in \mathcal{R}$.

Firstly, note that the bound is parallelizable as it splits as sum of functions of class-wise parameters w_k and each w_k can be optimized as,

$$\arg\min_{w_k} \quad \frac{\lambda}{2}\|w_k\|^2 - \sum_{i=1}^{N} y_{ik} w_k^\top x_i + \sum_{i=1}^{N} \log(1 + e^{w_k^\top x_i - a_i}) \tag{10}$$

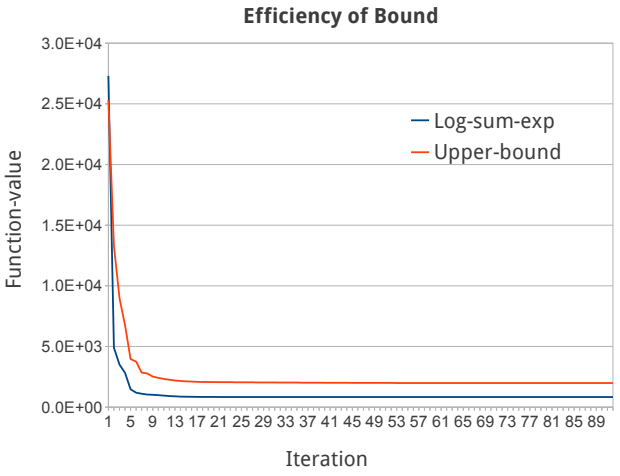

Figure 1. The figure shows the difference between the true minimum attained by the function and the upper bound using double majorization.

Secondly, it is differentiable and thirdly and importantly the upper bound is still convex. Although this bound seems to possess much better properties than the log-concavity bound, the only problem is that the bound is not tight enough. In our initial experiments we found that the gap between the log-sum-exp function and this upper bound is large. Figure 1 shows the gap between the true objective and the upper-bounded objective. The graph was generated by training two RMLR on the 20 news group dataset using the LBFGS optimization algorithm (Liu & Nocedal, 1989) and plotting the function-value after each iteration. The blue line shows the function-value using log-sum-exp and the red line shows the function-value by using the upper-bound. Since the gap is relatively large, we do not recommend using this bound.

3.4. Applicability of ADMM

Alternating direction method of multipliers (ADMM) is a relatively new technique that enables simple convex optimization problems to be easily parallelized. The key idea in ADMM is to introduce redundant linear constraints into the problem such that the optimization of the objective can be parallelized. In (Boyd et al., 2011), the authors show how ADMM can enable distributed computing for many machine learning models by either splitting across examples or splitting across parameters. Since we are particulary interested in parallelizing across class-parameters, we will look into the ADMM formulation for splitting across fea-

Table 1. Dataset Statistics

Dataset	# instances	#Leaf-labels	#Features	#Parameters	Train Size (approx)	Parameter Size (approx)
CLEF	10,000	63	80	5,040	3MB	40KB
NEWS20	11,260	20	53,975	1,079,500	11MB	4MB
LSHTC-small	4,463	1,139	51,033	227,760,279	5MB	911MB
LSHTC-large	93,805	12,294	347,256	4,269,165,264	129MB	17GB

tures. For a problem of the form

$$\min_{\mathbf{V}} \sum_{k=1}^{K} f_k(v_k) + g(\sum_{k=1}^{K} v_k)$$

the corresponding ADMM formulation is given by,

$$\min_{\mathbf{V},\mathbf{Z}} \sum_{k=1}^{K} f_k(v_k) + g(\sum_{k=1}^{K} z_k)$$
$$\text{subject to} \quad v_k - z_k = 0, \qquad k = 1,..K.$$

Here v_k are the parameters in the original problem and the z_k are the additional parameters introduced to enable distributed computing. f and g are both convex functions. The optimization problem is first solved by optimizing for each of the v_k in parallel. In the second step, to solve for z, instead of solving the problem as an optimization problem over k variables, we reduce it to an optimization problem over a single variable $\bar{z}$ by re-writing g as $g(K\bar{z})$. This neat trick enables efficient parallelization (refer (Boyd et al., 2011) pgs 56-57, 67-68)

However, this technique does not given any benefit when applied to RMLR model. For simplicity, consider a RMLR formulation with just one example x and $\lambda = 0$. The optimization problem for RMLR is given by

$$\min_{\mathbf{W}} \quad -\sum_{k=1}^{K} y_k w_k^\top x_i + \log\left(\sum_{k=1}^{K} \exp(w_k^\top x)\right)$$

To formulate the corresponding ADMM problem, it is clear that v_k needs to be set to class-level parameter w_k and $f_K(w_k) = -y_k w_k^\top x_i$ and g refers to the log-sum-exp function. Next we introduce redundant variables z_k such that $z_k = w_k^\top x_i$ for $k = 1,..K$ (note that the redundant constraints in ADMM should be linear). Now applying the ADMM formulation for N examples, we need to introduce $N \times K$ redundant variables i.e. z_{ik} variables which represent the prediction of training instance i w.r.t class k. This is given by,

$$\min_{\mathbf{W}} \quad -\sum_{k=1}^{K}\left(\sum_{i=1}^{N} y_k w_k^\top x_i\right) + \sum_{i=1}^{N} \text{log-sum-exp}(z_i)$$
$$\text{subject to} \quad w_k^\top x_i - z_{ik} = 0, \qquad i = 1,..N \quad k = 1,..K.$$

For solving, in each iteration, we need to solve K optimization problems of D (dimension) variables (i.e. for each w_k) and N optimization of K variables each (i.e. for each $\{z_{ik}\}_{k=1}^{K}$). There are several problems in this approach w.r.t RMLR objective. Firstly, the above requires much more computation than the log-concavity bound which requires solving only K optimization problems of D (dimension) variables (i.e. for each w_k) in each iteration. Secondly, the introduction of Z variable increases memory by $O(NK)$ as opposed to $O(N)$ variational parameters using the log-concavity bound . Lastly, as noted in (Boyd et al., 2011) (pg 17), ADMM exhibits very slow convergence properties. As we will see in the next section, it takes orders of magnitudes more computation (even after parallelization) to reach the same accuracy as log-concavity bound or LBFGS.

4. Experiments

Throughout our experiments, we consider 4 different datasets with increasing number of parameters to learn - CLEF [3], NEWS-20 [4], LSHTC-small, LSHTC-large [5]. An outline of the various characteristics of the dataset is given in Table 1.

We compare the following methods for distributed training of RMLR models,

1. **ADMM:** The alternating direction method of multipliers discussed in section 3.4. We tried multiple values of the ρ parameter (Boyd et al., 2011) and chose the one that offered the fastest convergence.
2. **LC:** The log-concavity bound in section 3.2.
3. **LBFGS:** The standard quasi-newton methods widely used to train logistic models (Liu & Nocedal, 1989). All the parameters of the model are optimized simultaneously. We use the previous 5 gradients to update the approximate hessian matrix. Furthermore, in order to make LBFGS as

[3] http://kt.ijs.si/DragiKocev/PhD/resources/doku.php?id=hmc_classification

[4] http://people.csail.mit.edu/jrennie/20Newsgroups/

[5] http://lshtc.iit.demokritos.gr/

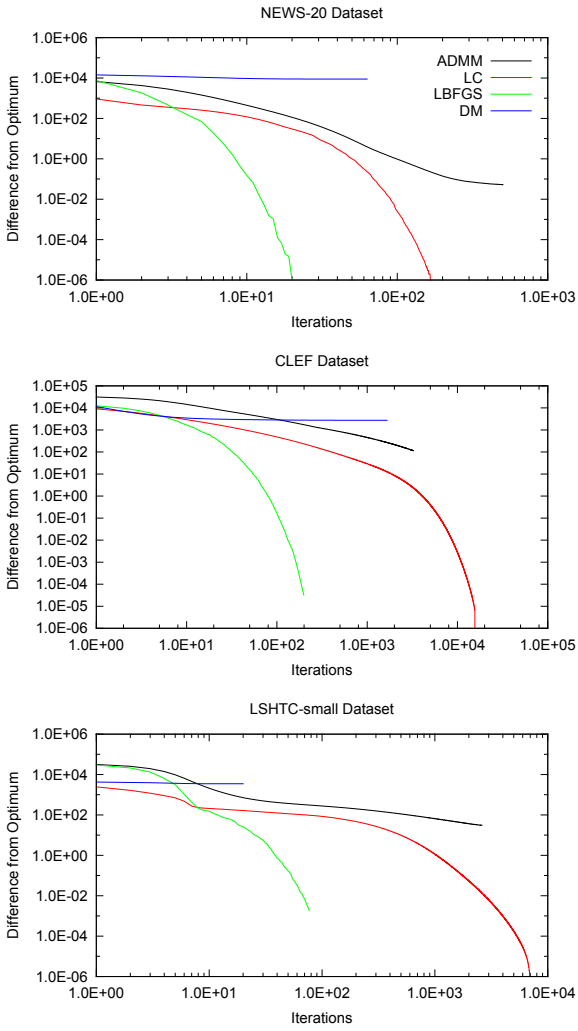

Figure 2. The difference from the true optimum as a function of number of iterations

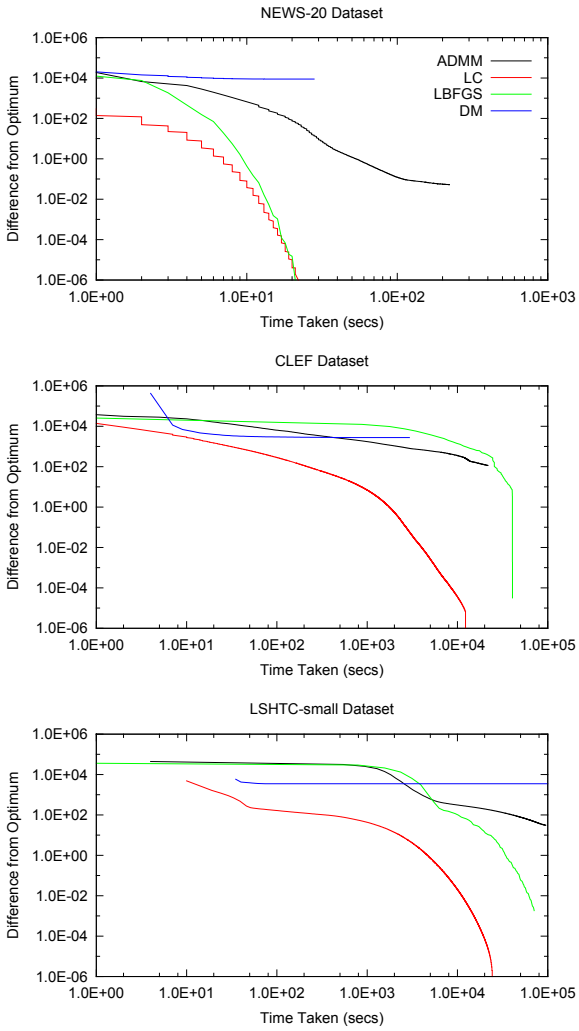

Figure 3. The difference from the true optimum as a function of time in seconds.

competitive as possible, the dot-product of an instance x_i with the class parameters w_k are computed in parallel (i.e. enables parallel computation of gradient).

4. **DM:** The Double Majorization in section 3.3.

For each inner problem in LC (2), DM (10) and ADMM we use LBFGS for optimization (other solvers can also be used, but LBFGS was chosen to maintain comparability). All methods were tested on a 48 core AMD Opteron Processor with 32GB RAM. For the largest LSHTC dataset, a map-reduce based Hadoop 20.2 cluster with 64 worker nodes with 8 cores and 16GB RAM (cluster has 220 mappers and 196 reducers) was used. Only the LC method could be scaled on this dataset. In each iteration of LC, the class parameters are parallely optimized in the map-phase and the variational parameters are updated in the reduce phase. Note that Hadoop in not a requirement, we chose to use it because the cluster could only be interfaced through Hadoop. In fact, other alternatives such as Peregrine [6], Haloop [7] , Twister [8] or non-hadoop alternatives such as MPI might be better choices as they can be customized for iterative parallelizable computations. The λ parameter for the datasets was selected using cross-validation using a one-versus rest logistic regression; for NEWS20 and CLEF, $\lambda = 1$, for LSHTC-small, $\lambda = .001$ and LSHTC, $\lambda = .01$.

Figures 2 show the difference from the optimal solution as the number of iteration increases for the first

[6]http://peregrine_mapreduce.bitbucket.org/
[7]http//code.google.com/p/haloop/
[8]http://www.iterativemapreduce.org/

three smaller datasets. In all the graphs, we see a common pattern : LBFGS takes the fewest number of iterations to converge to the optimal solution, followed by LC, ADMM and DM. This is not surprising because quasi-newton methods like LBFGS store some approximation of the Hessian which helps to achieve faster convergence than other methods. LC although being a block co-ordinate descent method seems to offer a much better convergence compared to ADMM or DM. ADMM as noted by the authors exhibits very slow convergence (Boyd et al., 2011) (pgs 6-7) to accurate solutions. DM does not even reach the optimal solution since the DM bound (3.3) is not tight.

Figures 3 show the difference from the optimal solution as a function of time taken. In this case, there is considerable shift in results. Among the four methods, only two of them seem to show reasonable convergence with time - LBFGS and LC. Comparing the 2 methods on the smallest dataset NEWS20, there does not seem to much difference between the them. But as the number of classes gets larger like the CLEF and LSHTC-small datasets, LC seems to perform significantly faster than LBFGS (and the other methods). Although LBFGS takes fewer iterations, each iteration is very computationally intensive and takes a long time, therefore each step of LBFGS is time consuming. This in contrast to LC where the cost per iteration is very cheap.

In fact on the largest LSHTC dataset, it is not even possible to run LBFGS due to the extreme memory requirements (17GB of parameters + 85GB of past gradient values need to be stored simultaneously in main memory). However, LC overcomes this difficulty by iteratively solving multiple subproblems and distributing the parameters across several computing units. Figure 4 shows the progress of LC on LSHTC as the number of iterations/time progresses. Most iterations of LC takes less than 6 minutes: around 3-4 minutes for optimizing the class-parameters and 2 minutes to update the variational parameters. This includes the time taken to start the hadoop job and transfer the parameters etc. Note that it is possible to train RMLR using ADMM or DM on this dataset, but we did not pursue this since neither of them showed reasonable convergence on the other datasets.

5. Conclusion and Future work

In this paper, we have explored multiple ways to train large-scale RMLR models using different ways of upper-bounding the log partition function. Our analysis and experiments have established that ADMM offers very slow convergence rate, DM has a loose upper-bound, LBFGS is too computationally and memory-intensive and LC being the most effective method. Unlike other methods, LC is able to successfully leverage distributed computing to achieve close to an order of magnitude reduction in training time.

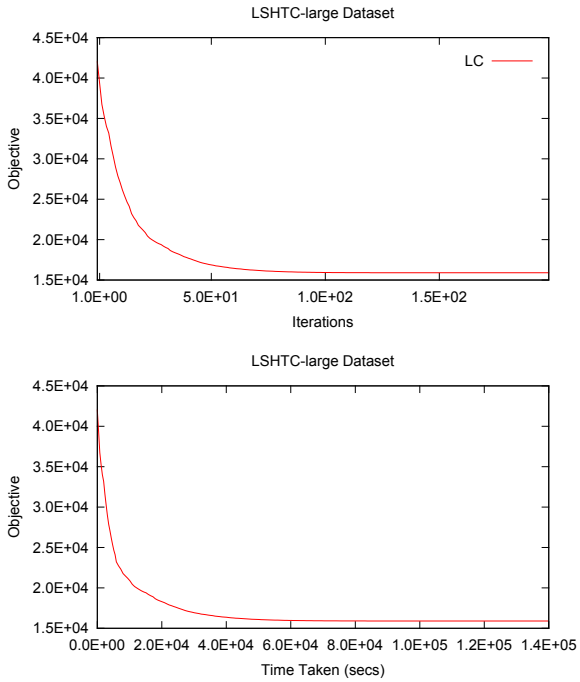

Figure 4. The progress of LC on the LSHTC dataset.

This work can be extended in many ways. In fact, after the publication of this work, we realized that there is a fully convex relaxation of the LC method that sidesteps the nonconvexity issues in OPT 2. This would enable a more thorough analysis of several aspects of the optimization such as the iteration complexity, the dependence on the strong convexity parameter λ, the effect of the correlation between the class-level parameters w_k etc. We plan to include these in an extended version of this work.

Acknowledgements

We thank the anonymous reviewers for their helpful suggestions and feedback. This work was supported in part by National Science Foundation (NSF) under grant IIS_1216282.

References

Blei, D.M. and Lafferty, J.D. Dynamic topic models. In *ICML*, pp. 113–120. ACM, 2006.

Bottou, L. Online learning and stochastic approxima-

tions. *On-line learning in neural networks*, 1998.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Compstat*, 2010.

Bouchard, G. Efficient bounds for the softmax function, applications to inference in hybrid models. 2007.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 2011.

Darroch, J.N. and Ratcliff, D. Generalized iterative scaling for log-linear models. *The annals of mathematical statistics*, 43(5):1470–1480, 1972.

Daumé III, H. Notes on cg and lm-bfgs optimization of logistic regression. 2004.

Holland, P.W. and Welsch, R.E. Robust regression using iteratively reweighted least-squares. *CSTM*, 1977.

Hsiung, K.L., Kim, S.J., and Boyd, S. Tractable approximate robust geometric programming. *Optimization and Engineering*, 9(2):95–118, 2008.

Kirkwood, B.R. et al. *Essentials of medical statistics.* Blackwell Scientific Publications, 1988.

Lin, C.J., Weng, R.C., and Keerthi, S.S. Trust region newton method for logistic regression. *The Journal of Machine Learning Research*, 9:627–650, 2008.

Liu, D.C. and Nocedal, J. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

Marlin, B., Khan, M.E., and Murphy, K. Piecewise bounds for estimating bernoulli-logistic latent gaussian models. In *ICML*, 2011.

Minka, T.P. A comparison of numerical optimizers for logistic regression. *Unpublished draft*, 2003.

Rust, R.T. and Zahorik, A.J. Customer satisfaction, customer retention, and market share. *Journal of retailing*, 69(2):193–215, 1993.

Schraudolph, N., Yu, J., and Günter, S. A stochastic quasi-newton method for online convex optimization. 2007.

Sha, F. and Pereira, F. Shallow parsing with conditional random fields. In *NAACL*, pp. 134–141. Association for Computational Linguistics, 2003.

Shanno, D.F. On broyden-fletcher-goldfarb-shanno method. *Journal of Optimization Theory and Applications*, 46(1):87–94, 1985.

Yu, H.F., Huang, F.L., and Lin, C.J. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1):41–75, 2011.

Zhang, T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML*, pp. 116. ACM, 2004.