Electronic Transactions on Numerical Analysis.
Volume 52, pp. 214–229, 2020.
Copyright © 2020, Kent State University.
ISSN 1068–9613.
DOI: 10.1553/etna_vol52s214

ADMM-SOFTMAX: AN ADMM APPROACH FOR MULTINOMIAL LOGISTIC REGRESSION[*]

SAMY WU FUNG[†], SANNA TYRVÄINEN[‡], LARS RUTHOTTO[¶], AND ELDAD HABER[§]

Abstract. We present ADMM-Softmax, an alternating direction method of multipliers (ADMM) for solving multinomial logistic regression (MLR) problems. Our method is geared toward supervised classification tasks with many examples and features. It decouples the nonlinear optimization problem in MLR into three steps that can be solved efficiently. In particular, each iteration of ADMM-Softmax consists of a linear least-squares problem, a set of independent small-scale smooth, convex problems, and a trivial dual variable update. The solution of the least-squares problem can be accelerated by pre-computing a factorization or preconditioner, and the separability in the smooth, convex problem can be easily parallelized across examples. For two image classification problems, we demonstrate that ADMM-Softmax leads to improved generalization compared to a Newton-Krylov, a quasi Newton, and a stochastic gradient descent method.

Key words. machine learning, nonlinear optimization, alternating direction method of multipliers, classification, multinomial regression

AMS subject classifications. 65J22, 90C25, 49M27

1. Introduction.

Multinomial classification seeks to assign the most likely label from a pre-defined set of three or more classes to all examples in a dataset. Classification is a key step in a wide range of applications such as data mining [32], neural signal processing [40], bioinformatics [35, 48], and text analysis [45]. The process can be described mathematically as a classifier (or hypothesis function) that maps each *input feature* to a vector in the unit simplex, whose components represent the predicted probabilities for each class.

In machine learning, the classifier is modeled as a fairly general parameterized function whose parameters (also called *weights*) are trained using a number of examples. Depending on the availability of labels for these data, one distinguishes between *supervised learning*, where labels are available for all examples, *unsupervised learning*, where only input features are provided, and *semi-supervised learning*, where only parts of the examples are labeled. Some well-known optimization problems arising in multinomial classification include multinomial logistic regression (MLR) [24] and support vector machines (SVMs) [37] for supervised learning, and k-nearest neighbors [23] for unsupervised learning. In this paper, we are primarily concerned with the efficient solution of the supervised learning problem arising in MLR.

When the dimensionality of the feature vector and the number of examples are large, a key challenge in MLR is the computational expense associated with solving a convex, smooth optimization problem. In short, the MLR problem consists of minimizing a (regularized) cross-entropy loss function used to compare the classifier's outputs to the given class probabilities. The classifier concatenates a softmax function, which maps a vector to the unit simplex, and an affine transformation whose weights are to be learned. Since the problem is smooth and convex, many standard optimization algorithms can be used to train the classifier, e.g., steepest

[*] Received July 11, 2019. Accepted February 2, 2020. Published online on April 17, 2020. Recommended by Marco Donatelli. S.W.F. and L.R.'s work was partially supported by the US National Science Foundation Awards DMS 1522599 and DMS 1751636.

[†] Department of Mathematics, UCLA, Los Angeles, California, USA (swufung@gmail.com).

[‡] Department of Mathematics, University of British Columbia, Vancouver, Canada (sannatyr@math.ubc.ca).

[§] Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia, Vancouver, Canada (haber@eoas.ubc.ca).

[¶] Department of Mathematics, Emory University, Atlanta, GA, USA (lruthotto@emory.edu).

descent [36], inexact-Newton or quasi-Newton methods [39], and perhaps the most commonly used in the machine learning community, stochastic gradient descent (SGD) [31, 53]. In practice, the performance of these methods can deteriorate close to the minimizer (manifesting, e.g., in a larger number of Conjugate Gradient (CG) steps in Newton-CG shown in Figure 4.4) and, particularly in the case of SGD, the limited potential for parallelization.

It is important to note that MLR is not equivalent to solving the optimization problem since an effective classifier must *generalize* well, i.e., it must assign accurate labels to instances not used during training. A known problem is *overfitting*, which occurs when the optimal weights fail to generalize. Hence, the optimization problem is not equivalent to the learning problem. To gauge the risk of overfitting, we partition the available training set into three subsets: the *training set*, which is used in the optimization, the *validation set*, which is used to gauge the generalization of the classifier and tune regularization and other parameters through cross-validation, and the *test set*, which is not used at all during training but used for the final assessment. As we will also demonstrate in our numerical experiments, the effect of overfitting in MLR is similar to the well-known semi-convergence in ill-posed inverse problems. This motivates the use of iterative regularization methods that have been very successful in inverse problems; see, e.g., [8, 14, 21, 26, 38].

Our contribution consists of a reformulation of the MLR optimization problem into a constrained problem and its approximate solution through the alternating direction method of multipliers (ADMM). Each iteration of the scheme requires the solution of a linear least-squares (LS) problem, a separable convex, smooth optimization problem, and a trivial dual update variable; see Section 3. The LS problem arising from the weights can be efficiently solved using direct or iterative solvers [17, 43]. Due to its separability, the smooth and convex problem arising from the second step can be solved in parallel for each example from the training set. We also provide all codes and examples used in this paper at https://github.com/swufung/ADMMSoftmax.

We test our method on two popular image classification problems: MNIST [34], a collection of hand-written digits, and CIFAR-10 [29], a collection of 32×32 color images divided into 10 classes. As is common in machine learning, we embed the original data into a higher-dimensional space before beginning the training process to improve the accuracy and generalization of the classifier. In particular, we propagate the MNIST dataset through a convolution layer with randomly chosen weights, which is also known as *extreme learning machines* (ELM); see Section 4.1.1 and [25]. For the CIFAR-10 dataset, we propagate the input data through a neural network which was previously trained on a similar dataset (also known as *transfer learning*). The pre-trained network we choose to use is AlexNet [29], which was trained on the ImageNet dataset [9], a dataset similar to CIFAR-10.

Our paper is organized as follows. In Section 2, we phrase MLR as an optimization problem and briefly review existing approaches for its solution. In Section 3, we present the mathematical formulation of the proposed ADMM-Softmax algorithm and discuss its computational costs and convergence properties. In Section 4, we compare our method to SGD and Newton-CG on the MNIST and CIFAR-10 datasets. Finally, we conclude our paper with a discussion in Section 5.

2. Multinomial logistic regression.

In this section, we review the mathematical formulation of multinomial logistic regression and discuss some related works. In training, we are given labeled data $(\mathbf{d}, \mathbf{c}) \in \mathbb{R}^{n_\mathrm{f}} \times \Delta_{n_\mathrm{c}}$ sampled from a typically unknown probability distribution. Here, $\mathbf{d}$ is the feature vector, n_f is the number of features (e.g., number of pixels in an image), $\mathbf{c}$ is the class label vector, and Δ_{n_c} denotes the unit simplex in $\mathbb{R}^{n_\mathrm{c}}$ where n_c is the number of classes. Since the class vector, $\mathbf{c}$, belongs to the unit simplex, we can interpret it as a discrete probability distribution.

The softmax classifier used to predict the class labels is given by

$$h_{\mathbf{W}}\left(\mathbf{d}\right) = \frac{\exp\left(\mathbf{W}\mathbf{d}\right)}{\mathbf{e}_{n_c}^\top \exp\left(\mathbf{W}\mathbf{d}\right)}. \tag{2.1}$$

Here, $h_{\mathbf{W}}\left(\mathbf{d}\right) \in \mathbb{R}^{n_c}$ is the predicted class label, $\mathbf{W} \in \mathbb{R}^{n_c \times n_f}$ is a weight matrix, $\mathbf{e}_{n_c} \in \mathbb{R}^{n_c}$ is a vector of all ones, and the exponential function is applied component-wise. To simplify our notation, we assume that (2.1) already contains a *bias* term that applies a constant shift to all transformed features and leads to an affine model. A simple way to incorporate the bias term is by appending the feature vector with a one and adding a new column to the weight matrix.

In the training, we seek to estimate weights $\mathbf{W}$ such that $h_{\mathbf{W}}\left(\mathbf{d}\right) \approx \mathbf{c}$ and that the predicted label (i.e., the index of the largest component of $h_{\mathbf{W}}\left(\mathbf{d}\right)$) matches the true label for all pairs $(\mathbf{d}, \mathbf{c})$. To quantify this, we use the expected cross-entropy loss function to measure the discrepancy between the true probabilities, $\mathbf{c}$, and the predicted probabilities, $h_{\mathbf{W}}\left(\mathbf{d}\right)$. In particular, we write the expected loss function as

$$\begin{aligned}\Phi(\mathbf{W}) &= \mathbb{E}\left[-\mathbf{c}^\top \log\left(h_{\mathbf{W}}\left(\mathbf{d}\right)\right)\right] = \mathbb{E}\left[-\mathbf{c}^\top \log\left(\frac{\exp\left(\mathbf{W}\mathbf{d}\right)}{\mathbf{e}_{n_c}^\top \exp\left(\mathbf{W}\mathbf{d}\right)}\right)\right] \\ &= \mathbb{E}\left[-\mathbf{c}^\top\mathbf{W}\mathbf{d} + \left(\mathbf{c}^\top\mathbf{e}_{n_c}\right)\left(\log\left(\mathbf{e}_{n_c}^\top \exp\left(\mathbf{W}\mathbf{d}\right)\right)\right)\right] \\ &= \mathbb{E}\left[-\mathbf{c}^\top\mathbf{W}\mathbf{d} + \log\left(\mathbf{e}_{n_c}^\top \exp\left(\mathbf{W}\mathbf{d}\right)\right)\right],\end{aligned}$$

where we use the fact that $\mathbf{c}^\top\mathbf{e}_{n_c} = 1$ since $\mathbf{c} \in \Delta_{n_c}$. The expected loss function consists of a linear term and a log-sum-exp term, and thus is convex and smooth [5]. In general, the loss couples all components of $\mathbf{W}$, which we will address with our proposed method.

To further aid generalization and avoid overfitting, it is common to add a Tikhonov regularization term and consider the following stochastic optimization problem of multinomial logistic regression

$$\operatorname*{arg\,min}_{\mathbf{W}} \left(\mathbb{E}\left[-\mathbf{c}^\top\mathbf{W}\mathbf{d} + \log\left(\mathbf{e}_{n_c}^\top \exp\left(\mathbf{W}\mathbf{d}\right)\right)\right] + \frac{\alpha}{2}\left\|\mathbf{L}\left(\mathbf{W} - \mathbf{W}_{\mathrm{ref}}\right)^\top\right\|_F^2\right), \tag{2.2}$$

where $\mathbf{L}$ is a regularization operator that may enforce, e.g., smoothness, and $\alpha > 0$ is a regularization parameter that balances the minimization of the loss and the regularity of the solution. Note here that $\mathbf{W}$ has dimensions $n_{\mathrm{c}} \times n_{\mathrm{f}}$ so that each row of $\mathbf{W}$ corresponds to the features of each class. Therefore, we transpose the weights in the regularization term so that we regularize the features rather than the classes of the weights. The matrix $\mathbf{W}_{\mathrm{ref}}$ is a reference solution around which the regularizer is centered and needs to be chosen by the user. During the training process, we monitor the effectiveness of the weights, for instances in the validation set to calibrate the regularization parameter and other *hyperparameters* using cross validation. After the training, we apply the classifier to the test dataset to quantify its potential to generalize.

Since we are primarily concerned with the splitting schemes, we focus the discussion on the quadratic regularizer. This choice also leads to a linear least-squares problem in our proposed scheme. A common alternative to the quadratic regularizer is ℓ_1 regularization, which is used to enforce sparsity of the weight matrix $\mathbf{W}$ and identify and extract the essential features in the data. Efficient approaches exist to address the non-smoothness introduced by this regularizer, e.g., interior-point methods [28], iterative shrinkage [20], and hybrid algorithms [44]; see also the survey by Yuan et al. [51]. These schemes can be incorporated into ADMM-Softmax.

Optimization methods for solving (2.2) can be broadly divided into two classes. Stochastic approximation (SA) methods such as stochastic gradient descent (SGD) and its variants, iteratively update $\mathbf{W}$ using gradient information computed with a single pair (or a small batch of pairs) randomly chosen from the training set [42]. Upon a suitable choice of the step size (also called *learning rate*) these methods can decrease the expected loss. In contrast, stochastic average approximation (SAA) schemes [27] approximate the expected loss with an empirical mean computed for a large batch containing N randomly chosen examples $(\mathbf{d}_1, \mathbf{c}_1), \ldots, (\mathbf{d}_N, \mathbf{c}_N)$, which leads to the deterministic optimization problem

$$\text{(2.3)} \quad \arg\min_{\mathbf{W}} \frac{1}{N} \sum_{j=1}^{N} \left[-\mathbf{c}_j^\top \mathbf{W}\mathbf{d}_j + \log\left(\mathbf{e}_{n_c}^\top \exp\left(\mathbf{W}\mathbf{d}_j \right) \right) \right] + \frac{\alpha}{2} \left\| \mathbf{L} \left(\mathbf{W} - \mathbf{W}_{\text{ref}} \right)^\top \right\|_F^2 .$$

Since the objective function is convex and differentiable, many standard gradient-based iterative optimization algorithms are applicable. The method proposed in this work is an SAA method, and our numerical experiments consider both SGD (an SA method) and several standard SAA methods for comparison. For an excellent review and discussion about the advantages and disadvantages of both approaches, we refer to [3].

In general, the objective functions in (2.2) and (2.3) are not separable, i.e., they couple all the components in $\mathbf{W}$. Due to the coupling, the Hessian cannot be partitioned and also changes in each iteration depending on the nonlinearity of the problem. The above reasons can render their solution computationally challenging, especially for high-dimensional data, and in the case of SAA methods, a large number of samples, N.

2.1. Related work. The wide-spread use of MLR in classification problems has led to the development of many numerical methods for its solution. A common thread is efforts to decouple the optimization problems (2.2) and (2.3) into several subproblems that can be solved efficiently and in parallel.

For example, [4, 18], use an upper bound based on the first-order concavity property of the log-function to decouple (2.3) into n_c subproblems associated with the different classes. The new optimization problem, however, is no longer convex. Other possible upper bounds that allow for a separable reformulation of MLR include quadratic upper bounds and a product of sigmoids. Detailed comparison of these and analytical solutions in a Bayesian setting can be found in [4]. In [18], Gopal and Yang use the concavity bound to solve multinomial logistic regression in parallel and show that their iterative optimization of the bounded objective converges to the same optimal solution as the unbounded original model. Related to the concavity bound, Fagan & Iyengar [12] and Raman et al. [41] use the convex conjugate of the negative log to reformulate the problem as a double-sum that can be solved iteratively with SA methods like SGD.

A method closely related to ours is [18], which reformulates the MLR problem (2.3) as a constrained optimization problem that decouples the linear and nonlinear terms of the cross-entropy loss and approximately solves the problem using an ADMM approach. Our method uses a similar splitting and parallelization scheme. As in [18], the existing optimization methods slightly outperform our scheme in terms of minimizing the expected loss over the training set; however, the obtained solutions of our scheme generalize better. Both approaches are inspired by the work of Boyd et al. [5], who solve sparse logistic regression problems in parallel by splitting it across features with ADMM.

Splitting techniques have also been applied to non-convex classification problems, e.g., the training of neural networks [46]. Here, the examples concentrate on binomial regression, which allows one to use a quadratic loss function and closed-form solutions for each iteration step. Another related approach to train neural networks is the method of auxiliary coordinates

(MAC) [6]. In MAC, new variables are introduced to decouple the problem. Unlike ADMM, however, MAC breaks the deep nesting (i.e., function compositions) in the objective function with the new parameters.

3. ADMM-Softmax.

In this section, we derive ADMM-Softmax, which is an SAA method for MLR. The main idea of our method is to introduce an auxiliary variable and associated constraint in (2.3) to obtain a separable objective function and then solve the problem approximately using an ADMM method. As common, the iterations of the ADMM method break down into easy-to-solve subproblems; see [5] for an excellent introduction to ADMM. In our case, we obtain a regularized linear least-squares problem, a separable convex, smooth optimization problem, and a trivial dual variable update. Efficient solvers exist for the first two subproblems.

By introducing global auxiliary variables $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N \in \mathbb{R}^{n_c}$, we reformulate (2.3) as

$$\begin{aligned} &\operatorname*{arg\,min}_{\mathbf{W},\mathbf{z}_1,\dots,\mathbf{z}_N} \ \frac{1}{N}\sum_{j=1}^{N}\left[-\mathbf{c}_j^\top \mathbf{z}_j + \log\left(\mathbf{e}_{n_c}^\top \exp\left(\mathbf{z}_j\right)\right)\right] + \frac{\alpha}{2}\left\|\mathbf{L}\left(\mathbf{W}-\mathbf{W}_{\text{ref}}\right)^\top\right\|_F^2 \\ &\qquad \text{s.t. } \ \mathbf{z}_j = \mathbf{W}\mathbf{d}_j, \quad j = 1,\dots,N. \end{aligned} \tag{3.1}$$

Note that this problem is equivalent to the SAA version of MLR in (2.3).

To solve (3.1) using ADMM-Softmax, we first consider the augmented Lagrangian

$$\begin{aligned} &\mathcal{L}_\rho\left(\mathbf{W}, \mathbf{z}_1, \dots, \mathbf{z}_N, \mathbf{y}_1, \dots, \mathbf{y}_N\right) = \\ &\quad \frac{1}{N}\sum_{j=1}^{N}\Bigg[-\mathbf{c}_j^\top \mathbf{z}_j + \log\left(\mathbf{e}_{n_c}^\top \exp\left(\mathbf{z}_j\right)\right) \\ &\qquad\qquad + \frac{\alpha}{2}\left\|\mathbf{L}\left(\mathbf{W}-\mathbf{W}_{\text{ref}}\right)^\top\right\|_F^2 + \mathbf{y}_j^\top\left(\mathbf{z}_j - \mathbf{W}\mathbf{d}_j\right) + \frac{\rho}{2}\left\|\mathbf{z}_j - \mathbf{W}\mathbf{d}_j\right\|_2^2\Bigg], \end{aligned}$$

where $\mathbf{y}_j \in \mathbb{R}^{n_c}$ is the Lagrange multiplier associated with the jth constraint, and $\rho > 0$ is a penalty parameter. The ADMM algorithm aims at finding the saddle point of the $\mathcal{L}_\rho$ by performing alternating updates. Denoting the values of the primal and dual variables at the kth iteration by $\mathbf{W}^{(k)}, \mathbf{z}_1^{(k)}, \dots, \mathbf{z}_N^{(k)}, \mathbf{y}_1^{(k)}, \dots, \mathbf{y}_N^{(k)}$, respectively, the scheme consists of the following three steps:

$$\begin{aligned} \mathbf{W}^{(k+1)} &= \operatorname*{arg\,min}_{\mathbf{W}} \ \mathcal{L}_\rho\left(\mathbf{W}, \mathbf{z}_1^{(k)}, \dots, \mathbf{z}_N^{(k)}, \mathbf{y}_1^{(k)}, \dots, \mathbf{y}_N^{(k)}\right), \\ \mathbf{z}_j^{(k+1)} &= \operatorname*{arg\,min}_{\mathbf{z}_j} \ \mathcal{L}_\rho\left(\mathbf{W}^{(k+1)}, \mathbf{z}_1^{(k)}, \dots, \mathbf{z}_{j-1}^{(k)}, \mathbf{z}_j, \mathbf{z}_{j+1}^{(k)} \dots, \mathbf{z}_N^{(k)}, \mathbf{y}_1^{(k)}, \dots, \mathbf{y}_N^{(k)}\right), \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j = 1,\dots,N, \\ \mathbf{y}_j^{(k+1)} &= \mathbf{y}_j^{(k)} + \rho\left(\mathbf{z}_j^{(k+1)} - \mathbf{W}^{(k+1)}\mathbf{d}_j\right), \quad j = 1,\dots,N. \end{aligned}$$

Note that in the second step we have used the fact that the Lagrangian does not couple the variables $\mathbf{z}_{j_1}$ and $\mathbf{z}_{j_2}$ for $j_1 \neq j_2$ to obtain N independent subproblems that can be solved in parallel. By introducing the scaled Lagrange multipliers $\mathbf{u}_j = (1/\rho)\mathbf{y}_j$ and dropping constant

Algorithm 1 ADMM-Softmax

```
Require: training data $(\mathbf{d}_j, \mathbf{c}_j)$,  $j = 1, \dots, N$
initialize $\mathbf{W}^{(1)}, \mathbf{z}_1^{(1)}, \dots, \mathbf{z}_N^{(1)}, \mathbf{u}_1^{(1)}, \dots, \mathbf{u}_N^{(1)}$, $\rho$, and $k = 1$
compute factorization or preconditioner for the Gram matrix (3.5)
while $k <$ maximum number of iterations do
    compute $\mathbf{W}^{(k+1)}$ by solving least-squares problem (3.2) (use pre-factorized coefficient matrix)
    compute $\mathbf{z}_1^{(k+1)}, \dots, \mathbf{z}_N^{(k+1)}$ by solving convex smooth problem (3.3) (potentially in parallel)
    compute $\mathbf{u}_j^{(k+1)} = \mathbf{u}_j^{(k)} + \left(\mathbf{z}_j^{(k+1)} - \mathbf{W}\mathbf{d}_j^{(k+1)}\right)$,   $j = 1, \dots, N$,
    if stopping criteria (3.4) is satisfied then
        break
    end if
    $k \leftarrow k + 1$
end while
```

terms in the respective optimization problems, the steps simplify to

$$\mathbf{W}^{(k+1)} = \arg\min_{\mathbf{W}} \frac{\rho}{2} \sum_{j=1}^{N} \left(\left\| \mathbf{z}_j^{(k)} - \mathbf{W}\mathbf{d}_j + \mathbf{u}_j^{(k)} \right\|_2^2 \right) + \frac{\alpha}{2} \left\| \mathbf{L} \left(\mathbf{W} - \mathbf{W}_{\text{ref}} \right)^\top \right\|_F^2 \tag{3.2}$$

$$\mathbf{z}_j^{(k+1)} = \arg\min_{\mathbf{z}_j} -\mathbf{c}_j^\top \mathbf{z}_j + \log \left(\mathbf{e}_{n_c}^\top \exp\left(\mathbf{z}_j\right) \right) + \frac{\rho}{2} \left\| \mathbf{z}_j - \mathbf{W}^{(k+1)}\mathbf{d}_j + \mathbf{u}_j^{(k)} \right\|_2^2, \quad j = 1, \dots, N, \tag{3.3}$$

$$\mathbf{u}_j^{(k+1)} = \mathbf{u}_j^{(k)} + \left(\mathbf{z}_j^{(k+1)} - \mathbf{W}^{(k+1)}\mathbf{d}_j \right), \quad j = 1, \dots, N,$$

The first subproblem (3.2) is a linear least-squares problem whose coefficient matrix is independent of k (see Section 3.1), and the second subproblem (3.3) decouples into N independent problems, each of which is a convex, smooth optimization problem and involves n_c variables; see Section 3.2.

Let us note in passing that a different regularization in the original optimization problem (2.3) would only impact the least-squares subproblem in (3.2), and the formulation of the **z**-steps in (3.3) would remain unchanged. Therefore, one can use any existing algorithms to efficiently solve least-square problems with different types of regularization terms.

To terminate the ADMM iteration, a common stopping criteria is described in [5], where the norms of the primal and dual residuals are defined as

$$\left\| \mathbf{r}^{(k+1)} \right\|_2 = \sum_{j=1}^{N} \left\| \mathbf{r}_j^{(k+1)} \right\|_2 = \sum_{j=1}^{N} \left\| \mathbf{z}_j^{(k+1)} - \mathbf{W}^{(k+1)}\mathbf{d}_j \right\|_2, \text{ and}$$

$$\left\| \mathbf{s}^{(k+1)} \right\|_2 = \sum_{j=1}^{N} \left\| \mathbf{s}_j^{(k+1)} \right\|_2 = \sum_{j=1}^{N} \left\| \rho \operatorname{vec} \left(\left(\mathbf{z}_j^{(k+1)} - \mathbf{z}_j^{(k)} \right) \mathbf{d}_j^\top \right) \right\|_2,$$

respectively, where for any matrix $\mathbf{X}$, the operator vec($\mathbf{X}$) returns its vectorized form. The stopping criterion is satisfied when

$$\begin{aligned}\left\|\mathbf{r}^{(k+1)}\right\|_2 \le \epsilon_{\mathrm{pri}}^{(k)} &= \sqrt{N n_{\mathrm{c}}}\epsilon_{\mathrm{abs}} \\ &\quad + \epsilon_{\mathrm{rel}}\max\left\{\left\|\mathbf{z}_1^{(k)}\right\|_2, \dots, \left\|\mathbf{z}_N^{(k)}\right\|_2, \left\|\mathbf{W}^{(k)}\mathbf{d}_1\right\|_2, \dots, \left\|\mathbf{W}^{(k)}\mathbf{d}_N\right\|_2\right\} \\ \left\|\mathbf{s}^{(k+1)}\right\|_2 \le \epsilon_{\mathrm{dual}}^{(k)} &= \sqrt{N n_{\mathrm{c}}}\epsilon_{\mathrm{abs}} + \epsilon_{\mathrm{rel}}\max\left\{\left\|\mathbf{u}_1^{(k)}\right\|_2, \dots, \left\|\mathbf{u}_N^{(k)}\right\|_2\right\},\end{aligned} \tag{3.4}$$

where $\epsilon_{\mathrm{rel}} > 0$ and $\epsilon_{\mathrm{abs}} > 0$ are the relative and absolute tolerances chosen by the user. For a summary of the proposed scheme, we refer to Algorithm 1.

3.1. W-update.

Letting

$$\begin{aligned}\mathbf{Z}^{(k)} &= \left[\mathbf{z}_1^{(k)}\ \mathbf{z}_2^{(k)} \dots \mathbf{z}_N^{(k)}\right] \in \mathbb{R}^{n_{\mathrm{c}} \times N}, \\ \mathbf{U}^{(k)} &= \left[\mathbf{u}_1^{(k)}\ \mathbf{u}_2^{(k)} \dots \mathbf{u}_N^{(k)}\right] \in \mathbb{R}^{n_{\mathrm{c}} \times N}, \text{ and} \\ \mathbf{D} &= [\mathbf{d}_1\ \mathbf{d}_2 \dots \mathbf{d}_N] \in \mathbb{R}^{n_{\mathrm{f}} \times N},\end{aligned}$$

we can rewrite the $\mathbf{W}$-update (3.2) as

$$\mathbf{W}^{(k+1)} = \arg\min_{\mathbf{W}} \frac{\rho}{2}\left\|\mathbf{Z}^{(k)} - \mathbf{W}\mathbf{D} + \mathbf{U}^{(k)}\right\|_F^2 + \frac{\alpha}{2}\left\|\mathbf{L}\left(\mathbf{W} - \mathbf{W}_{\mathrm{ref}}\right)^\top\right\|_F^2,$$

which amounts to solving n_c independent linear least-squares (i.e., one for each row in $\mathbf{W}$). This is equivalent to solving normal equations

$$\mathbf{A}_{\alpha,\rho}\mathbf{W}^\top = \rho\mathbf{D}\left(\mathbf{Z} + \mathbf{U}\right)^\top + \alpha\mathbf{L}\mathbf{L}^\top\mathbf{W}_{\mathrm{ref}}^\top, \tag{3.5}$$

where $\mathbf{A}_{\alpha,\rho} = \left(\rho\mathbf{D}\mathbf{D}^\top + \alpha\mathbf{L}\mathbf{L}^\top\right)$. Noting that the coefficient matrix in (3.5) is not iteration-dependent, depending on the number of features, the matrix can be factorized once (e.g., using Cholesky applied to the normal equations or a thin-QR applied to the original least-squares problem [17]) and its inverse can be quickly applied, leading to trivial solves throughout the optimization. We also note that this is only one possible approach for solving (3.5), and that large-scale problems can be addressed using iterative methods; see, e.g., [1, 2, 13, 15, 16, 17, 43]. As the performance of most iterative methods can be improved using pre-conditioning, we can compute a preconditioner (e.g., incomplete Cholesky factorization) in an off-line phase and re-use it in the ADMM iterations.

3.2. z-update.

Each of the objective functions in the $\mathbf{z}_j$-updates in (3.3) can be written as

$$\Psi(\mathbf{z}_j) = -\mathbf{c}_j^\top\mathbf{z}_j + \log\left(\mathbf{e}_{n_{\mathrm{c}}}^\top \exp\left(\mathbf{z}_j\right)\right) + \frac{\rho}{2}\left\|\mathbf{z}_j - \mathbf{z}_{j,\mathrm{ref}}\right\|_2^2,$$

where $j = 1, \dots, N$, and $\mathbf{z}_{j,\mathrm{ref}} = \mathbf{W}^{(k+1)}\mathbf{d}_j - \mathbf{u}_j^{(k)}$. In our numerical experiments, we solve these n_{c}-dimensional convex, smooth optimization problems using a Newton method. The gradient and Hessian of the objective are

$$\nabla_{\mathbf{z}_j}\Psi(\mathbf{z}_j) = -\mathbf{c}_j + \frac{\exp\left(\mathbf{z}_j\right)}{\mathbf{e}_{n_{\mathrm{c}}}^\top \exp\left(\mathbf{z}_j\right)} + \rho\left(\mathbf{z}_j - \mathbf{z}_{j,\mathrm{ref}}\right) \in \mathbb{R}^{n_{\mathrm{c}}},$$

and

$$\nabla^2_{\mathbf{z}_j}\Psi(\mathbf{z}_j) = \frac{\operatorname{diag}\left(\exp\left(\mathbf{z}_j\right)\right)}{\mathbf{e}_{n_c}^\top \exp\left(\mathbf{z}_j\right)} - \frac{\exp\left(\mathbf{z}_j\right)\exp\left(\mathbf{z}_j\right)^\top}{\left(\mathbf{e}_{n_c}^\top \exp\left(\mathbf{z}_j\right)\right)^2} + \rho\operatorname{diag}\left(\mathbf{z}_j\right) \in \mathbb{R}^{n_c\times n_c}, \tag{3.6}$$

respectively. Here, for a generic vector $\mathbf{v}$, the operator diag$(\mathbf{v})$ returns a diagonal matrix with $\mathbf{v}$ along its diagonal. For examples with many classes, the Newton direction can be approximated using an iterative solver, e.g., the preconditioned conjugate gradient method (PCG). In this case, we do not build the Hessian in (3.6) explicitly; instead, we implement its action on a vector in order to save memory.

3.3. Computational costs and convergence. A computationally challenging step in ADMM-Softmax is solving the least-squares problem (3.2), for which there is a myriad of efficient solvers; see, e.g., [17] for an extensive review. Noting that the coefficient matrix, $\mathbf{A}_{\alpha,\rho}$ in (3.5) is iteration-independent, it could be factorized in the off-line phase with, e.g., Cholesky or thin QR, and we can trivially solve least-squares throughout the optimization.

The $\mathbf{z}$-updates in (3.3) can be solved in independently and in parallel for each $j = 1, \ldots, N$. In our experiments, we solve the $\mathbf{z}$-updates using Newton-CG. Assuming n_p workers are available, the cost for each Hessian matrix-vector product from (3.6) is in the order of about $\mathcal{O}\left(\frac{N}{n_\mathrm{p}} n_\mathrm{c}^3\right)$ FLOPS per worker, leading to very fast computations of the global variable update. If $n_\mathrm{p} = N$, that is, if we have a worker for each example, the cost per worker is in the order of $\mathcal{O}(n_\mathrm{c}^3)$ FLOPS. We note that the number of class labels is usually relatively small compared to the number of features (in our experiments, for instance, $n_\mathrm{c} = 10$), making (3.3) negligible when solved in parallel.

As for convergence, it has been shown that the ADMM algorithm converges linearly for convex problems with an existing solution regardless of the choice $\rho > 0$ [11]. If the subproblems (3.2) or (3.3) are solved inexactly, ADMM still converges under additional conditions. In particular, the sequences

$$\mu^{(k)} = \left\|\mathbf{W}^{(k+1)} - \mathbf{W}^{(k)}\right\|_F \quad \text{and} \quad \gamma_j^{(k)} = \left\|\mathbf{z}_j^{(k+1)} - \mathbf{z}_j^{(k)}\right\|_2, \quad j = 1, \ldots, N,$$

must be summable. This allows us to solve the subproblems iteratively, especially for large-scale problems. More details can be found in [11].

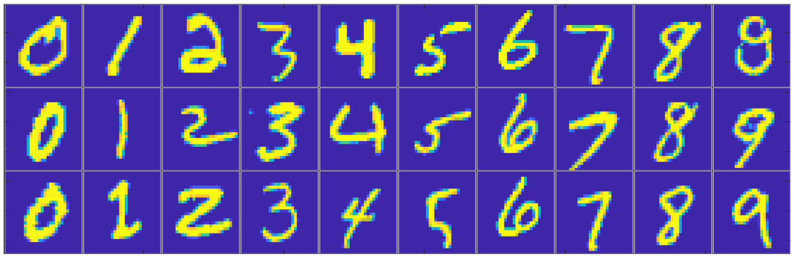

FIG. 3.1. *Example of hand-written images obtained from the MNIST dataset.*

4. Numerical experiments.

In this section, we demonstrate the potential of our proposed ADMM-Softmax method using the MNIST and CIFAR-10 datasets. For both datasets, we first compare three algorithms: The SAA methods ADMM-Softmax, Newton-CG, and ℓ-BFGS and the SA method SGD. We then study the behavior of ADMM-Softmax and its dependence

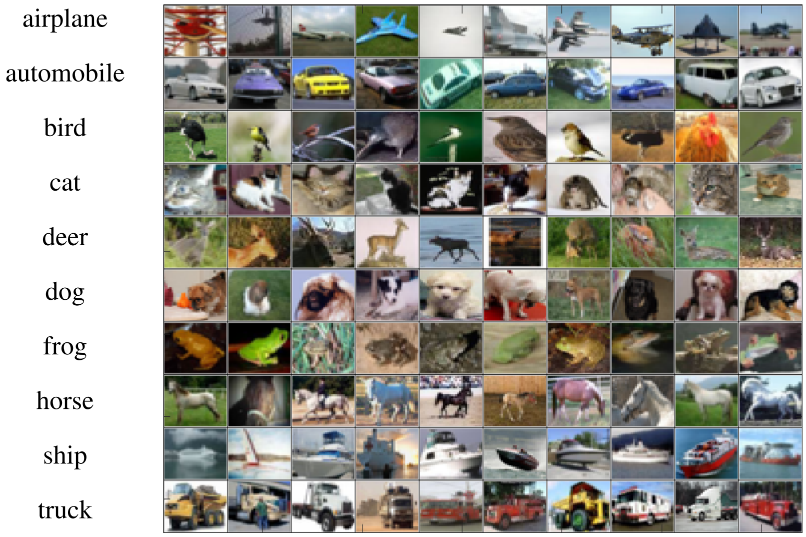

FIG. 3.2. *Example images for the CIFAR-10 dataset.*

.

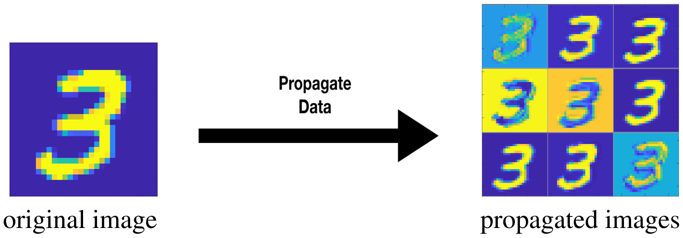

FIG. 3.3. *Feature transformation of a single image from the MNIST dataset. Here, we apply a convolution layer with randomly chosen weights to increase the dimensionality of each example image increases from* $28^2 = 784$ *image (left) to* $n_f = (3 \times 28)^2 = 7056$ *(right).*

on the penalty and regularization parameter. We illustrate the challenges of Newton-CG in MLR by a comparison of the PCG performance at the first and final nonlinear iteration. We perform our experiments in MATLAB using the Meganet deep learning package [49] and we provide our code at `https://github.com/swufung/ADMMSoftmax`. We perform all of our experiments on a shared memory computer operating Ubuntu 14.04 with 2 Intel Xeon E5-2670 v3 2.3 GHz CPUs using 12 cores each, and a total of 128 GB of RAM.

4.1. Setup.

4.1.1. MNIST. The MNIST database consists of 60,000 grey-scale hand-written images of digits ranging from 0 to 9; see [33, 34]. Here, we set 40,000 examples for training our digit-recognition system, 10,000 are used for validation, and the remaining 10,000 is used as testing data. Each image has 28 × 28 pixels. A few randomly chosen examples are shown in Figure 3.1.

Since the MNIST dataset is not linearly separable, we embed the original features into a higher-dimensional space obtained by a nonlinear transformation to the original variables; this procedure creates new features, which improves the performance of multinomial logistic regression. In particular, for each image, we obtain nine transformed images by using a convolution layer with 3×3 randomly chosen filters, i.e.,

$$\mathbf{d}_{j,\text{prop}} = \tanh(\mathbf{K}\mathbf{d}_j) \in \mathbb{R}^m, \tag{4.1}$$

where all 9 vertical blocks of $\mathbf{K} \in \mathbb{R}^{9n_\text{f} \times n_\text{f}}$ are 2D convolution operators. This process is also known as *extreme learning machines* [25]. Here, we assume periodic boundary conditions which render each block to be a block-circulant matrix with circulant blocks (BCCB) [22]. The transformed features have dimensions $m = 9 \cdot n_\text{f} + 1 = 7057$. Expanding the features increases the effective rank of the feature matrix $\mathbf{D}$ and increase the capacity of the classifier. An illustration of the propagated features is shown in Figure 3.3.

We compare four algorithms: our proposed ADMM-Softmax, Newton-CG, ℓ-BFGS, and SGD. In SGD, we use Nesterov momentum with minibatch size 300, and learning rate $l_r = 10^{-1}$. Here, we choose the learning rate and minibatch sizes by performing a grid-search on $[10^{-12}, 10^3]$ and $[1, 500]$ to maximize the performance on the validation set, respectively. The initial learning rate grid-search is done logarithmically, whereas the minibatch sizes are uniformly spaced. In Newton-CG, we set a maximum number of 20 inner CG iterations per Newton iteration, with CG tolerance of 10^{-2}. In ℓ-BFGS, we store the 10 most recent vectors used to approximate the action of the Hessian on a vector at each iteration, and solve the inner system with the same settings as in Newton-CG.

For the ADMM-Softmax, we perform a grid-search on $[10^{-16}, 10^3]$ and report the ρ that led to the best validation accuracy - in this case, $\rho = 10^{-7}$. To solve the LS system, we compute a QR factorization in the off-line phase, which for this experiment took about 20 seconds. To solve (3.3), we use the Newton-CG method from the Meganet package using a maximum of 100 iterations with gradient norm stopping tolerance of 10^{-8}, and initial condition $\mathbf{z}_j^{(k)}$ (i.e., *warm start*). Since we do not parallelize step (3.3) in our implementation, our input for (3.3) is given by $\mathbf{Z}^{(k)} = [\mathbf{z}_1^{(k)}, \dots, \mathbf{z}_N^{(k)}] \in \mathbb{R}^{n_\text{c} \times N}$, and the resulting Hessian is block diagonal. As a result, we solve the inner Newton system using a maximum of 50 inner iterations and stopping tolerance of 10^{-8}. Note that if we solved (3.3) in parallel using N workers, we would not need more than $n_\text{c} = 10$ CG iterations, as each individual Hessian would have size $n_\text{c} \times n_\text{c}$.

For all three methods, we use a discrete Laplace operator as the regularization operator $\mathbf{L}$ with $\alpha = 10^{-6}$ to enforce smoothness of the images. This choice of regularization operator is motivated by the interpretation of the rows of $\mathbf{W} \in \mathbb{R}^{n_c \times n_f}$ as discrete images [19, Section 5.2]. The probability of a sample $\mathbf{y} \in \mathbb{R}^{n_f}$ belonging to the k^{th} class depends on the inner product $\mathbf{w}_k^\top \mathbf{y}$, which occurs inside the softmax function. Thus, we wish to obtain weights that are insensitive to small displacements in the images, i.e., weights that favor spatially smooth parameters. We set reference weights to $\mathbf{W}_\text{ref} = \mathbf{0}$ and choose the α that led to the best validation accuracy for the Newton-CG method.

4.1.2. CIFAR-10.

The CIFAR-10 dataset [29] consists of 60,000 RGB images of size 32×32 that are divided into ten classes. As in MNIST, we split the data as follows: 40,000 for training, 10,000 as validation, and 10,000 as testing data. The images belong to one of the following ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

For this dataset, we increase the feature space by propagating the input features through a neural network (AlexNet [30]), which was trained on the ImageNet dataset [9] using MATLAB's deep neural networks toolbox - this procedure is also known as *transfer learning*. The main difference to the extreme learning machine that we used for MNIST is that instead of propagating through a random hidden layer as in (4.1), we propagate through a network whose layers have already been trained on a *similar* dataset to that of CIFAR-10. In this case, we extract the features from the *pool5* layer in AlexNet. For this dataset, each example $\mathbf{d}_j$

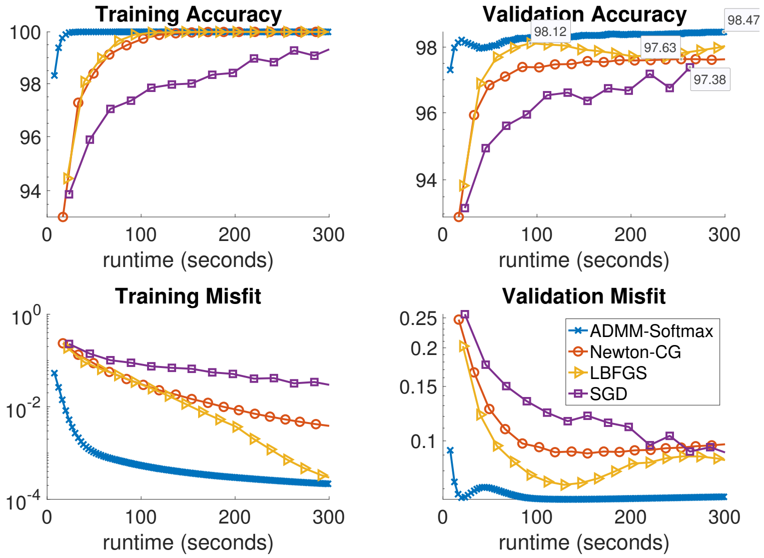

FIG. 4.1. *For the MNIST dataset, we visualize the accuracy of the classifier (top row) and associated values of the loss functions (bottom row) computed using the training set (left column) and validation set (right column) at each iteration of the training algorithms. The x-axes show the runtime in seconds. It can be seen that the proposed ADMM-Softmax outperforms the other methods both on the training and the validation set.*

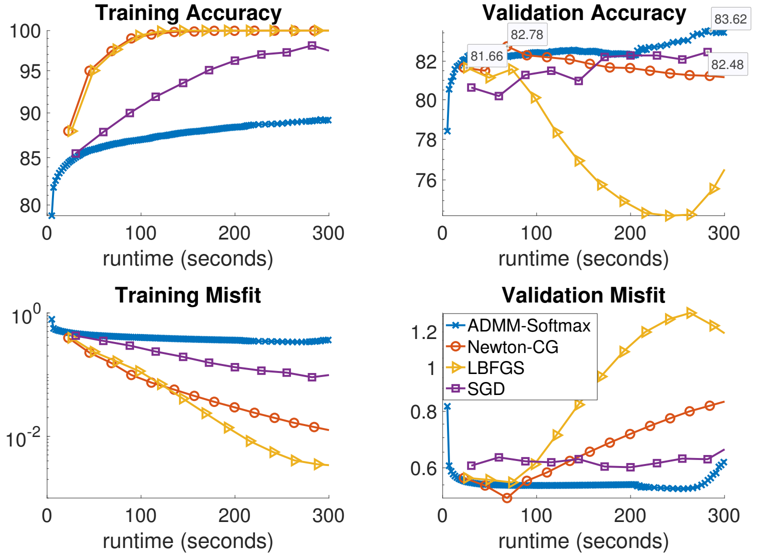

FIG. 4.2. *For the CIFAR dataset, we visualize the accuracy of the classifier (top row) and associated values of the loss functions (bottom row) computed using the training set (left column) and validation set (right column) at each iteration of the training algorithms. The x-axes show the runtime in seconds. In this test, the ADMM-Softmax method shows inferior performance on the training dataset but leads to a slight improvement of both the loss and classification accuracy on the validation dataset.*

is an RGB image of dimension $32 \times 32 \times 3$ and the propagated data $\mathbf{d}_{j,\mathrm{prop}}$ has dimension $6 \times 6 \times 256$.

As for the optimization, we maintain the same parameter choice as in the MNIST dataset except for the following. For SGD, we repeat our grid-search for the learning rate and minibatch sizes as in MNIST and report the parameters that led to the best results. This leads to a smaller learning rate of $l_r = 10^{-2}$ but the same minibatch size of 300. For ADMM-Softmax, we choose the penalty parameter as $\rho = 8 \times 10^{-12}$ in the same manner as in Section 4.1.1. In this case, the propagated features no longer correspond to RGB images. As before, we use the Laplace operator as the regularization operator with $\mathbf{W}_{\text{ref}} = \mathbf{0}$ and chose the α that led to the best results for Newton-CG, in this case, $\alpha = 10^{-1}$.

4.2. Comparison. In Figure 4.1 and Figure 4.2, we show the training and validation accuracies and misfits for all algorithms applied to the MNIST dataset and CIFAR-10 dataset, respectively. The former demonstrates the effectiveness of the optimization scheme, while the latter gauges the classifier's ability to generalize, which is the main concern in MLR. Since the computational work required for each algorithm per iteration is different, we report the runtimes of each algorithm and provide an equal computational budget of 300 seconds to all methods and instances. As can be seen, we are able to afford many more ADMM-Softmax iterations than the other algorithms within the 300-second budget (noting again that the off-line factorizations took about 20 seconds). This is because the LS-problem and $\mathbf{z}$-updates took on average 0.9 and 0.6 seconds, respectively, per iteration for the MNIST dataset (despite not using any parallelization). Similarly, they took 1.6 and 0.5 seconds on average, respectively, per iteration for the CIFAR-10 dataset. This is dependent on the computational platform and implementation, but we conducted a serious effort to optimize the standard methods, while not realizing the parallelization potential of ADMM-Softmax. We also provide all codes needed to replicate our experiment. To compare the performance of each algorithm, we pick the weights in the iteration containing the highest validation accuracy for each respective algorithm and use them to classify the testing dataset. We show these results in Table 4.1.

It is important to note that although the training process is formulated as an optimization problem, we are not necessarily solving an optimization problem, since this may not mean better generalization. Instead, we wish for an algorithm that leads to the best validation dataset. For instance, in Figure 4.2, the Newton-CG method is the most effective algorithm at reducing the training misfit, however, this makes the algorithm overfit to the training set, leading to worse performance on the validation dataset. This can be seen in the semi-convergence behavior of the validation misfits in Figure 4.2, and is a reason why SGD and ADMM are popular methods in the machine learning community. This situation is analogous to that of solving ill-posed inverse problems iteratively, where the objective is not necessarily choosing the parameters that best fit the (potentially noisy) data. Finally, we note that we did not use any parallelization in any of these experiments; however, we expect that further speedups of the ADMM-Softmax method can be achieved by computing the $\mathbf{z}$-update (3.3) in parallel, particularly for larger datasets.

TABLE 4.1
Validation and testing misfits and accuracies for ADMM-Softmax, Newton-CG, ℓ-BFGS, and SGD (row-wise) for the MNIST and CIFAR-10 datasets. In each case, we show the accuracy computed using the iterate with the best performance on the validation dataset.

	MNIST				CIFAR-10			
	validation		testing		validation		testing	
	misfit	accuracy	misfit	accuracy	misfit	accuracy	misfit	accuracy
ADMM-Softmax	0.066	98.47 %	0.092	97.74 %	0.570	83.62 %	0.590	82.90 %
Newton-CG	0.095	97.63 %	0.130	96.46 %	0.514	82.78 %	0.534	81.58 %
ℓ-BFGS	0.078	98.12 %	0.093	97.34 %	0.570	81.66 %	0.584	81.03 %
SGD	0.093	97.38 %	0.113	96.56 %	0.627	82.48 %	0.664	80.79 %

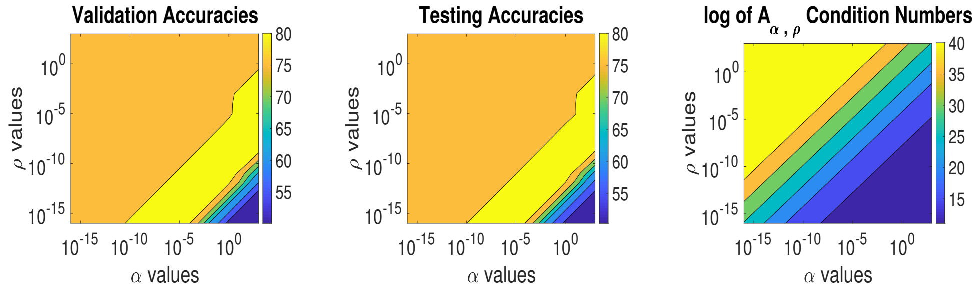

FIG. 4.3. *Validation accuracies (left), and testing accuracies (right), and condition numbers of $\mathbf{A}_{\alpha,\rho}$ defined in* (3.5) *(right) of ADMM-Softmax with respect to different ρ (y-axis) and α (x-axis) values for the CIFAR-10 dataset. Here, α and ρ are sampled from* $[10^{-16}, 10^{3}]$.

4.3. Parameter dependence. We study the dependence of ADMM-Softmax on the penalty and regularization parameters ρ and α, where for brevity, we use only the CIFAR-10 dataset since it is the more challenging dataset; as can be seen in Section 4.2. In Figure 4.3, we show the validation and testing accuracies for different values of ρ and α sampled from $[10^{-16}, 10^{-3}]$. As in Section 4.2, the weights that led to the highest validation accuracy were chosen to classify the testing dataset. The accuracy behavior is similar for the testing and validation datasets; thus, the validation dataset gives us a good indication of the generalizability of our classifier during the optimization. On the right, we show the condition numbers of $\mathbf{A}_{\alpha,\rho}$ (see (3.5)). As expected, smaller values of α lead a more ill-conditioned $\mathbf{A}_{\alpha,\rho}$, however, this can be remedied with sufficiently small ρ.

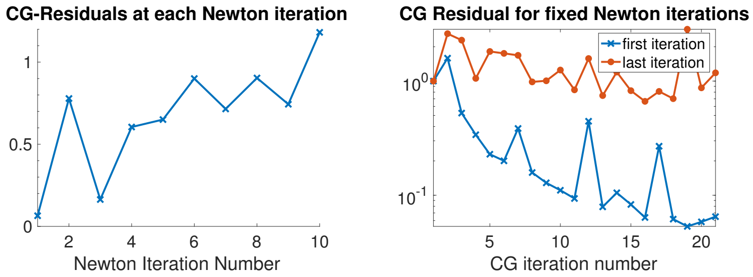

FIG. 4.4. *We exemplarily show the deterioration of the performance of the CG iteration at the first and final Newton-CG iteration. The left subplot shows the relative residual of the solution returned by CG at each Newton iteration. The right subplot shows the relative residuals at each CG step for the first and final Newton iteration. Both plots demonstrate that the performance of CG tends to be worse at later Newton-CG iterations.*

4.4. Deterioration of Newton-CG Performance. As a final experiment, we exemplarily show that the performance of the inner iteration of Newton-CG deteriorates as we approach the global minimum. We have observed this phenomenon in many numerical experiments. Recall that the performance of CG depends mainly on the clustering of the eigenvalues of the Hessian of the objective function in (2.3), which is iteration-dependent; for an excellent discussion of CG, see [43]. We note that the Hessians are too large for us to reliably compute a full spectral decomposition in a reasonable amount of time, so as one indicator, we plot the residuals of the CG-solver after each Newton iteration on the left of Figure 4.4. The plot shows that CG is less effective in later Newton-CG iterations. We also show the relative residuals for each inner CG iteration at the first and last Newton-CG iteration. Here, a decrease in performance is also evident as the last Newton iteration is quicker to stall than the first Newton iteration. ADMM-Softmax circumvents this problem as the Hessians in the $\mathbf{z}$-update are much smaller and easier to solve.

5. Conclusion.

In this paper, we present ADMM-Softmax, a simple and efficient algorithm for solving multinomial logistic regression (MLR) problems arising in classification. To this end, we reformulate the traditional MLR problem consisting of an unconstrained optimization into a constrained optimization problem with a separable objective function. The new problem is solved by the alternating direction method of multipliers (ADMM), whose iteration consists of three simpler steps, i.e., a linear least-squares, a large number of independent convex, smooth optimization problems, and a trivial dual variable update. ADMM-Softmax allows the use of standard method for each of these substeps. In our experiments, we solve the resulting least-squares problems using a direct solver with a pre-computed factorization, and the nonlinear problems using a Newton method; see Section 3.

Our method is also inspired by the successful applications of ADMM to ℓ_1-regularized linear inverse problems, also known as *lasso* [47, 52] and *basis pursuit* [7]. Here, ADMM breaks the *lasso* problem into two subproblems: one containing a linear least-squares problem, and the other containing a decoupled nonlinear, non-smooth term, which amends a closed-form solution given by soft thresholding [10]. Our problem can be similarly divided into a linear least-squares problem and a set of decoupled smaller problems involving a nonlinear cross-entropy loss minimization. One distinction is that our second substep is solved using a Newton scheme.

Our numerical results show improved generalization when compared to Newton-CG, ℓ-BFGS, and SGD for the MNIST and CIFAR-10 datasets. Further benefits are to be expected for large datasets where parallelization is necessary. We note that better accuracies, especially for the CIFAR-10 dataset, could be achieved if we re-train the parameters of pre-trained AlexNet [50] (rather than keeping them fixed). To this end, our method can accelerate block-coordinate algorithms that alternate between updating the network weights and the classifier. This is a direction of our future work. Our results can be reproduced using the codes provided at `https://github.com/swufung/ADMMSoftmax`.

Acknowledgments. This material is supported by the U.S. National Science Foundation (NSF) through awards DMS 1522599 and DMS 1751636. We also thank the Isaac Newton Institute (INI) for Mathematical Sciences for the support and hospitality during the programme on generative models, parameter learning, and sparsity.

REFERENCES

[1] M. Benzi, *Preconditioning techniques for large linear systems: a survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[2] M. Benzi and M. Tůma, *A robust incomplete factorization preconditioner for positive definite matrices*, Numer. Linear Algebra Appl., 10 (2003), pp. 385–400.

[3] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM Rev., 60 (2018), pp. 223–311.

[4] G. Bouchard, *Efficient bounds for the softmax function, applications to inference in hybrid models*, presentation at the workshop for approximate Bayesian inference in continuous/hybrid systems at NIPS-07, Whistler, Canada, Dec. 2007.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Found. Trends Mach. Learn., 3 (2010), pp. 1–122.

[6] M. Á. Carreira-Perpiñán and W. Wang, *Distributed optimization of deeply nested systems*, in Proceedings of the 17th International Conference on Artificial Intelligence and Statistics Reykjavik, S. Kaski and J. Corander, eds., vol. 33 of Proceedings of Machine Learning Research, PMLR, 2014, pp. 10–19.

[7] S. S. Chen, D. L. Donoho, and M. A. Saunders, *Atomic decomposition by basis pursuit*, SIAM Rev., 43 (2001), pp. 129–159.

[8] J. Chung, J. G. Nagy, and D. P. O'Leary, *A weighted-GCV method for Lanczos-hybrid regularization*, Electron. Trans. Numer. Anal., 28 (2007/08), pp. 149–167.
http://etna.mcs.kent.edu/vol.28.2007-2008/pp149-167.dir/pp149-167.pdf

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-fei, *Imagenet: A large-scale hierarchical image database*, in 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE Conference Proceedings, Los Alamitos, 2009, pp. 248-255.

[10] D. L. Donoho and I. M. Johnstone, *Adapting to unknown smoothness via wavelet shrinkage*, J. Amer. Statist. Assoc., 90 (1995), pp. 1200–1224.

[11] J. Eckstein and D. P. Bertsekas, *On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators*, Math. Programming, 55 (1992), pp. 293–318.

[12] F. Fagan and G. Iyengar, *Unbiased scalable softmax optimization*, Preprint on arXiv, 2018.
https://arxiv.org/abs/1803.08577

[13] S. Gazzola, P. C. Hansen, and J. G. Nagy, *IR Tools: a MATLAB package of iterative regularization methods and large-scale test problems*, Numer. Algorithms, 81 (2019), pp. 773–811.

[14] S. Gazzola, S. Noschese, P. Novati, and L. Reichel, *Arnoldi decomposition, GMRES, and preconditioning for linear discrete ill-posed problems*, Appl. Numer. Math., 142 (2019), pp. 102–121.

[15] S. Gazzola, P. Novati, and M. R. Russo, *On Krylov projection methods and Tikhonov regularization*, Electron. Trans. Numer. Anal., 44 (2015), pp. 83–123.
http://etna.mcs.kent.edu/vol.44.2015/pp83-123.dir/pp83-123.pdf

[16] S. Gazzola, E. Onunwor, L. Reichel, and G. Rodriguez, *On the Lanczos and Golub-Kahan reduction methods applied to discrete ill-posed problems*, Numer. Linear Algebra Appl., 23 (2016), pp. 187–204.

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, 1996.

[18] S. Gopal and Y. Yang, *Distributed training of large-scale logistic models*, in Proceedings of the 30th International Conference on Machine Learning Atlanta, S. Dasgupta and D. McAllester, eds., vol. 28 of Proceedings of Machine Learning Research, PMLR, pp. 289–297.

[19] E. Haber and L. Ruthotto, *Stable architectures for deep neural networks*, Inverse Problems, 34 (2018), Art. 014004, 22 pages.

[20] E. T. Hale, W. Yin, and Y. Zhang, *Fixed-point continuation for l_1-minimization: methodology and convergence*, SIAM J. Optim., 19 (2008), pp. 1107–1130.

[21] P. C. Hansen, *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*, SIAM, Philadelphia, 2005.

[22] P. C. Hansen, J. G. Nagy, and D. P. O'leary, *Deblurring Images: Matrices, Spectra, and Filtering*, SIAM, Philadelphia, 2006.

[23] T. Hastie and R. Tibshirani, *Discriminant adaptive nearest neighbor classification and regression*, in Advances in Neural Information Processing Systems 8 (NIPS 1995) , D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, eds., MIT Press, Cambridge, 1996, pp. 409–415.

[24] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer, New York, 2001.

[25] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, *Extreme learning machine: theory and applications*, Neurocomputing, 70 (2006), pp. 489–501.

[26] M. E. Kilmer and D. P. O'Leary, *Choosing regularization parameters in iterative methods for ill-posed problems*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1204–1221.

[27] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, *The sample average approximation method for stochastic discrete optimization*, SIAM J. Optim., 12 (2001/02), pp. 479–502.

[28] K. Koh, S.-J. Kim, and S. Boyd, *An interior-point method for large-scale l1-regularized logistic regression*, J. Mach. Learn. Res., 8 (2007), pp. 1519–1555.

[29] A. Krizhevsky, *Learning multiple layers of features from tiny images*, Tech. Report, Department of Computer Science, University of Toronto, 2009.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Proceedings of the 25th International Conference on Neural Information Processing Systems Vol. 1, F Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., Curran Associates, Red Hook, 2012, pp. 1097–1105.

[31] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, *On optimization methods for deep learning*, in Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11, L. Getoor and T. Scheffer, eds., Omnipress, Madison, 2011, pp. 265–272.

[32] T. H. Le, T. T. Tran, and L. K. Huynh, *Identification of hindered internal rotational mode for complex*

chemical species: A data mining approach with multivariate logistic regression model, Chemometrics Intell. Lab. Sys., 172 (2018), pp. 10–16.

[33] Y. LECUN, L. BOTTOU, Y. BENGIO, AND P. HAFFNER, *Gradient-based learning applied to document recognition*, Proc. IEEE, 86 (1998), pp. 2278–2324.

[34] Y. LECUN, C. CORTES, AND C. J. BURGES, *The MNIST database of handwritten digits*, Online recource, 1998. http://yann.lecun.com/exdb/mnist/

[35] J. LIAO AND K.-V. CHIN, *Logistic regression for disease classification using microarray data: model selection in a large p and small n case*, Bioinformatics, 23 (2007), pp. 1945–1951.

[36] R. MALOUF, *A comparison of algorithms for maximum entropy parameter estimation*, in Proceedings of the 6th Conference on Natural Language Learning COLING-02, Vol. 20, Association for Computational Linguistics, 2002, pp. 1–7.

[37] F. MELGANI AND L. BRUZZONE, *Classification of hyperspectral remote sensing images with support vector machines*, IEEE Trans. Geosci. Remote Sens., 42 (2004), pp. 1778–1790.

[38] J. G. NAGY, K. PALMER, AND L. PERRONE, *Iterative methods for image deblurring: a Matlab object-oriented approach*, Numer. Algorithms, 36 (2004), pp. 73–93.

[39] J. NOCEDAL AND S. J. WRIGHT, *Numerical Optimization*, Springer, New York, 1999.

[40] L. PARRA, C. SPENCE, AND P. SAJDA, *Higher-order statistical properties arising from the nonstationarity of natural signals*, In Advances in Neural Information Processing Systems 13 (NIPS 2000), T.K. Leen, T.G. Dietterich, and V. Tresp., eds., NIPS Proceedings, 2001, pp. 786–792.

[41] P. RAMAN, S. MATSUSHIMA, X. ZHANG, H. YUN, AND S. V. N. VISHWANATHAN, *DS-MLR: exploiting double separability for scaling up distributed multinomial logistic regression*, Preprint on arXiv/CoRR, 2016. http://arxiv.org/abs/1604.04706

[42] H. ROBBINS AND S. MONRO, *A stochastic approximation method*, Ann. Math. Statistics, 22 (1951), pp. 400–407.

[43] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.

[44] J. SHI, W. YIN, S. OSHER, AND P. SAJDA, *A fast hybrid algorithm for large-scale l1-regularized logistic regression*, J. Mach. Learn. Res., 11 (2010), pp. 713–741.

[45] M. TADDY, *Distributed multinomial regression*, Ann. Appl. Stat., 9 (2015), pp. 1394–1414.

[46] G. TAYLOR, R. BURMEISTER, Z. XU, B. SINGH, A. PATEL, AND T. GOLDSTEIN, *Training neural networks without gradients: a scalable ADMM approach*, in Proceedings of the 33rd International Conference on Machine Learning, M. F. Balcan and K. Q. Weinberger, eds., JMLR, New York, 2016, pp. 2722–2731.

[47] R. TIBSHIRANI, *Regression shrinkage and selection via the lasso*, J. Roy. Statist. Soc. Ser. B, 58 (1996), pp. 267–288.

[48] Y. TSURUOKA, J. MCNAUGHT, J. TSUJII, AND S. ANANIADOU, *Learning string similarity measures for gene/protein name dictionary look-up using logistic regression*, Bioinformatics, 23 (2007), pp. 2768–2774.

[49] XTRACTOPEN, *Meganet*, Matlab package, 2018. https://github.com/XtractOpen,

[50] J. YOSINSKI, J. CLUNE, Y. BENGIO, AND H. LIPSON, *How transferable are features in deep neural networks?*, in Proceedings of the 27th International Conference on Neural Information Processing Systems NIPS'14, Vol. 2, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds., MIT Press, Cambridge, 2014, pp. 3320–3328.

[51] G.-X. YUAN, K.-W. CHANG, C.-J. HSIEH, AND C.-J. LIN, *A comparison of optimization methods and software for large-scale L1-regularized linear classification*, J. Mach. Learn. Res., 11 (2010), pp. 3183–3234.

[52] M. YUAN AND Y. LIN, *Model selection and estimation in regression with grouped variables*, J. R. Stat. Soc. Ser. B Stat. Methodol., 68 (2006), pp. 49–67.

[53] T. ZHANG, *Solving large scale linear prediction problems using stochastic gradient descent algorithms*, in Proceedings of the 21st International Conference on Machine Learning, ICML '04, ACM, New York, 2004, pp. 116–124.