A Maximally Split and Adaptive Relaxed Alternating Direction Method of Multipliers for Regularized Extreme Learning Machines

Zhangquan Wang 1(✉), Shanshan Huo 2, Xinlong Xiong 3, Ke Wang 1,4,* and Banteng Liu 1

1 College of Information and Technology, Zhejiang Shuren University, Hangzhou 310015, China; 600389@zjsru.edu.cn (Z.W.); 3110102872@zju.edu.cn (B.L.)
2 School of Computer Science and Artificial Intelligence, Changzhou University, Changzhou 213164, China
3 College of Information Engineering, Zhejiang University of Technology, Hangzhou 310014, China
4 State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China
* Correspondence: wangke1992@zju.edu.cn

Abstract: One of the significant features of extreme learning machines (ELMs) is their fast convergence. However, in the big data environment, the ELM based on the Moore–Penrose matrix inverse still suffers from excessive calculation loads. Leveraging the decomposability of the alternating direction method of multipliers (ADMM), a convex model-fitting problem can be split into a set of sub-problems which can be executed in parallel. Using a maximally splitting technique and a relaxation technique, the sub-problems can be split into multiple univariate sub-problems. On this basis, we propose an adaptive parameter selection method that automatically tunes the key algorithm parameters during training. To confirm the effectiveness of this algorithm, experiments are conducted on eight classification datasets. We have verified the effectiveness of this algorithm in terms of the number of iterations, computation time, and acceleration ratios. The results show that the method proposed by this paper can greatly improve the speed of data processing while increasing the parallelism.

Keywords: extreme learning machines; alternating direction method of multipliers; matrix calculation; convex optimization

MSC: 68T09

1. Introduction

The extreme learning machine (ELM) has been extensively applied in many areas [1] due to its fast learning ability and satisfactory generalization performance. The regularized ELM (RELM) [2] is an extended variant of the standard ELM [3] which improves the generalization performance and stability of ELMs by adding a regularization term in the loss function [4]. However, the dimension and the volume of data have increased significantly with the development in big data. When the number of training samples and the number of hidden layer nodes are especially large, the size of the output matrix of the ELM model will be particularly large. Therefore, the calculation of the ELM based on the Moore–Penrose matrix inverse requires humongous storage and calculations, significantly increasing the computational complexity of the ELM.

To address the above problems, several enhanced ELMs were proposed. By decomposing the data matrix into a set of smaller block matrices, Wang et al. [5] adopted a clustering technique with a message-passing interface to train the block-matrix-based ELM in parallel with the aim of improving computing efficiency. Liu et al. [6] proposed a Spark-distributed parallel computing mechanism to achieve a parallel transformation of ELMs. Chen et al. [7] used a clustering technique with GPUs to parallelize ELMs. Based on the Spark framework, Duan et al. [8] improved the learning speed of the ELM when processing large-scale data by...
dividing the dataset. All the methods discussed above focus on computation schemes of the RELM using parallel or distributed hardware structures. However, the matrix-inversion-based (MI-based) solution process has low efficiency and high computational complexity, leading to low convergence [9]. Therefore, all the methods discussed above cannot solve the problem of the low efficiency of RELMs in the big data scenario.

The alternating direction method of multipliers (ADMM) is a powerful computational framework for separable convex optimization. It has been extensively applied in many fields owing to its fast processing speed and convergence performance [10]. Wang et al. [11] used the ADMM to solve the center selection problem in fault-tolerant radial-basis-function networks. Wei et al. [12] applied the ADMM to neural networks to solve the problem of slow training for large-scale data. Wang et al. [13] applied the ADMM to SVMs to achieve distributed learning by splitting the training samples. Luo et al. [14] used the decomposability of the ADMM; then, the regularized least-squares (RLS) problem of the RELM can be split into a set of sub-problems that can be executed in parallel to achieve the purpose of improving computation efficiency. Li et al. [15] used the ADMM to solve the predictive control problem of a distributed model, which enables the model to have a fast-response ability. Xu et al. [16] applied the ADMM to a quantized recurrent neural network language training model into an optimization problem to improve its convergence speed.

One of the main problems of the classical ADMM is the convergence speed. In general, the numerical performance of the ADMM largely depends on an effective solution to the sub-problems—there can be several different sub-problem splitting representations in practical applications. Thus, a generalization of the $N$-block ADMM is needed because the classical ADMM algorithm is only suitable for solving two-block convex optimization problems, and the sub-problems structure cannot be fully utilized.

To further improve the ADMM convergence speed and generalization performance, several extended variants of the ADMM were presented, including the generalized ADMM [17–20] and the relaxed ADMM (RADMM) [21]. Lai et al. achieved fast convergence by using a novel relaxation technique to modify the ADMM to have an ADMM with a highly parallelized structure. Based on the RADMM, Xiaoping et al. [22] proposed a maximally split and relaxed ADMM (MS-RADMM) by considering splitting model coefficients to improve convergence and parallelism. Su et al. [23] introduced a binary splitting operator into the ADMM, and the optimal solution of the original problem is obtained through the iterative calculation of the intermediate operators to achieve the purpose of improving the convergence speed. Ma et al. [24] used an MS-RADMM with a highly parallel structure to optimize a 2D FIR filter, and a practical scheme for algorithm parameter setting was provided. Hou et al. [25] utilized a tunable step-size algorithm to accelerate MS-ADMM convergence speed. However, the convergence speed of the ADMM largely depends on the choice of parameters in the iterative process. For this reason, we propose an adaptive parameter selection method which uses the improved Barzilai–Borwein spectral gradient method to automatically tune the algorithm parameters to achieve an optimal convergence speed.

For the implementation of the MS-RADMM, we propose an adaptive parameter selection method for joint tuning of the penalty and relaxation parameters. Our main contributions are as follows:

(1) Improving Global Convergence: To improve the global convergence of the algorithm, a non-monotonic Wolfe-type strategy is introduced into the memory gradient method. The global optimal solution is achieved by combining the iteration information of current and past multiple function points.

(2) Solving Sub-problem: To improve the convergence speed of the algorithm, the Barzilai–Borwein spectral gradient method is optimized by adding step-size selection constraints to simplify the computational complexity of the MS-RADMM sub-problems.
2. Fundamentals of the RELM and the ADMM

With an increase in the volume and complexity of datasets, the size of training samples \(N\) and the number of the hidden nodes \(L\) are very large. As such, the MI-based solutions require enormous memory space and suffer from excessive computational loads. To address these challenges, the ADMM is used to handle the convex model-fitting problem of the ELM.

2.1. RELM Method

As a training framework for solving single hidden-layer neural networks [26], the ELM has a good learning speed and generalization performance. For an \(m\)-category classification problem, assuming that the training sample is \(x\) and number of hidden-layer nodes is \(L\), the ELM model output is given by

\[
f_L(x) = \sum_{i=1}^{L} h_i(x) \beta = H(x) \beta = T, \tag{1}
\]

where

\[
h_i(x) = g(w_i x + s_i), \tag{2}
\]

where \(H(x) = [h_1(x), h_2(x), \ldots, h_L(x)]\) is the hidden-layer output matrix, \(w_i, s_i\) are the input weight and bias of the \(i\)th hidden nodes, \(g(\cdot)\) is the activation function, \(\beta\) is the output weight matrix, and \(T\) denotes the target output matrix of the network.

The actual performance of the ELM depends on the number of neurons in the hidden-layer. If the number of neurons is too small, the extracted information is insufficient, and it is hard to generalize and reflect the inherent disciplines of the data. If the number is too large, the network structure is too complex, thus reducing the generalization performance.

To further improve the generalization performance and the stability, regularization theory is imported into the ELM to minimize the training error and the norm of the output weight matrix \(\beta\) [27–29]. The RELM solves for the output weight \(\beta\) in the following RLS problem

\[
\min_{\beta} \frac{1}{2} ||H\beta - T||_F^2 + \frac{1}{2} \mu ||\beta||_F^2, \tag{3}
\]

where \(||\cdot||_F\) denotes the Frobenius norm, and \(\mu > 0\) is a regularizer that controls the tradeoff between the loss function and a regularization term.

However, the MI-based RELM leads to an excessive computational load, particularly in problems concerning high-dimensional data. An effective way to solve large-scale data processing problems is through parallel or distributed optimization methods. The ADMM is a powerful technique for large-scale convex optimization.

2.2. ADMM for Convex Optimization

As a computational framework for solving constrained optimization problems, the ADMM achieves good performance on convergence speed and parallel structures. The ADMM [30] decomposes a large global problem into multiple local sub-problems, and the solution of the global problem is obtained through coordinating the solutions of the sub-problems. The following convex model-fitting problem is studied:

\[
\min_f(A x - b) + r(x), \tag{4}
\]

where \(A = [a_1, a_2, \ldots, a_L] \in R^{N \times L}, a_i \in R^N\), \(A\) represents the data matrix, \(x = [x_1, x_2, \ldots, x_L]^T \in R^{L \times m}, x_i \in R^m\) denotes the convex model coefficient vector, \(b \in R^{N \times m}\) is the target output vector, \(f(\cdot)\) means a convex loss function, and \(r(\cdot)\) is a regularization term.

By defining equality constraints \(z_i = a_i x_i\), the model-fitting problem (3) can be transformed into

\[
\min_f \left( \sum_{i=1}^{L} z_i - b \right) + \sum_{i=1}^{L} r(x_i). \tag{5}
\]
The augmented Lagrangian of problem (5) is

\[
L_{\rho}(x, z, \lambda) = f(\sum_{i=1}^{L} z_i - b) + \sum_{i=1}^{L} r(x_i) + \sum_{i=1}^{L} \lambda_i^f (a_i x_i - z_i) + \frac{\rho}{2} \sum_{i=1}^{L} ||a_i x_i - z_i||_2^2,
\]

where \(\rho > 0\) is the penalty parameter, and \(\lambda_i \in \mathbb{R}^{N \times m}\) is the dual variable.

The ADMM uses the Gauss–Seidel iteration method [31] to minimize the augmented Lagrangian function of optimized variables \(x\) and \(z\) and updates the dual variable \(\lambda\) according to a multiplier method. The iterative solution process of the model-fitting problem is easily obtained as

\[
\begin{align*}
\alpha^{k+1} &= \arg\min_{\alpha_i} \left\{ g(x_i) + \frac{\rho}{2} ||a_i x_i - z_i^k + \lambda_i^k||_2^2 \right\} \\
z^{k+1} &= \arg\min_{z_i} \left\{ f(\sum_{i=1}^{L} z_i - b) \right\} + \left\{ \frac{\rho}{2} \sum_{i=1}^{L} ||a_i x_i^k - z_i + \lambda_i^k||_2^2 \right\} \\
\lambda_i^{k+1} &= \lambda_i^k + \rho(a_i x_i^{k+1} - z_i^{k+1})
\end{align*}
\]

The global optimal solution is obtained by alternately updating variables \(x\) and \(z\) [32].

3. Maximally Split and Adaptive Relaxed ADMM

3.1. Maximally Split and Relaxed ADMM

The numerical performance of the ADMM largely depends on the efficient solving of sub-problems [33]. A maximally split technique and a relaxation technique are used to speed up the ADMM convergence [34].

The MS-ADMM splits the model-fitting problem into multiple univariate sub-problems flexibly with a reasonable scale. It reconstructs the method, based on matrix operations, ensuring that there is only one scalar component in each sub-problem. This gives the MS-ADMM an ideal highly parallel structure. By considering the \(L\)-partition ADMM [35], matrix \(A\) is simplified to a column vector \(a_i\) and the vector coefficient \(x\) is simplified to a scalar coefficient \(x_i\). These scalar characteristics play an important role in improving the parallel computing efficiency and the highly parallel structure of the MS-ADMM.

On the basis of the MS-ADMM, the MS-RADMM [36] is acquired by adopting a relaxation technique. It reconstructs the convergence conditions; past iterations are considered in the next iteration, which makes the MS-RADMM have linear convergence. By enlarging the equality constraint residuals \(a_i x_i - z_i = 0\), \(z_i^k\) is replaced with \(a z_i + (1 - a) a_i x_i^{k+1}\), \(z_i\) with \(a z_i + (1 - a) a_i x_i^{k+1}\), and \(z_i^{k+1}\) with \(\alpha z_i^{k+1} + (1 - a) a_i x_i^{k+1}\) in (7). The MS-RADMM is expressed as

\[
\begin{align*}
\alpha_i^{k+1} &= \arg\min_{\alpha_i} \left\{ g(x_i) + \frac{\rho}{2} ||a_i (x_i - x_i^k)||_2^2 \right\} - \left\{ ||a_i (z_i^k - a_i x_i^{k+1} - \lambda_i^k||_2^2 \right\} \\
z_i^{k+1} &= \arg\min_{z_i} \left\{ f(\sum_{i=1}^{L} z_i - b) \right\} + \left\{ \frac{\rho}{2} \sum_{i=1}^{L} ||z_i - a_i x_i^{k+1} + \lambda_i^k||_2^2 \right\} \\
\lambda_i^{k+1} &= \lambda_i^k + \alpha \rho(a_i x_i^{k+1} - z_i^{k+1})
\end{align*}
\]

where \(\alpha > 0\) is the relaxation parameter, magnifying the equality constraint residuals.

3.2. Scalars MS-ARADMM

The efficiency of the MS-RADMM depends strongly on the choice of the penalty and relaxation parameters. A suitable parameter selection scheme is key to improving the computational efficiency of the MS-RADMM.

We propose an adaptive parameter selection method for the MS-RADMM and obtain the MS-ARADMM. The MS-ARADMM allows for automatic tuning of the key algorithm.
parameters to improve the convergence speed. The convergence is measured by using the primal and dual residuals, defined as

\[ \gamma_k = ax_k - z_k, \]  
\[ d_k = \rho a^T(z_k - z_{k-1}). \] 

From the perspective of the convergence principle, when the algorithm approaches the optimal solution, \( \gamma_k, d_k \) residuals are close to zero. The specific termination conditions are shown as

\[ \|\gamma_k\| \leq \epsilon_{tol} \max\{\|Hx\|, \|z\|\}, \] 
\[ \|d_k\| \leq \epsilon_{tol} \max\{\|\lambda\|\}, \]

where \( \epsilon_{tol} \) represents the stop tolerance and is a constant; the specific value can be set according to the actual error range. Considering the time cost in the experiment process, the stop tolerance is set to \( 10^{-3} \) in this paper, and the setting of the stop tolerance is only related to the accuracy of the error and does not depend on the dataset.

### 3.2.1. Spectral Adaptive Step-Size Rule

The spectral adaptive step-size rule is derived by studying the close relationship between the RADMM \cite{37} and the relaxed Douglas–Rachford splitting (DRS) \cite{38}.

For problem (5), assume that a local linear model of \( \partial f(x) \) and \( \partial r(x) \) at iteration \( k \) is given by

\[ \partial f(x) = \theta_k x + \psi_k, \] 
\[ \partial r(x) = \gamma_k x + \phi_k, \] 

where \( \theta_k > 0, \gamma_k > 0 \) are the local curvature estimates of \( f \) and \( r \), respectively. \( \psi, \phi \) are constants.

According to the equivalence of the RADMM and the DRS, the linear model is fitted to the gradient of the target by using DRS theory for problem (13). In order to obtain the optimal step-size with zero residuals on the model problem, such that \( f(x) + r(x) \) residuals are zero, the following needs to be satisfied:

\[ \alpha_k = 1 + \frac{1 + \theta_k \gamma_k^2}{\theta_k + \gamma_k} \] 
\[ \rho_k = \arg\min_{\rho_k} \frac{1 + \theta_k \gamma_k^2 \rho_k^2}{(\theta_k + \gamma_k)\rho_k} = \frac{1}{\sqrt{\theta_k \gamma_k}}. \] 

We can readily find the optimal relaxation parameter under the optimal penalty parameter condition

\[ \alpha_k = 1 + \frac{1 + \theta_k \gamma_k^2}{(\theta_k + \gamma_k)\rho_k} = 1 + \frac{2\sqrt{\theta_k \gamma_k}}{\theta_k + \gamma_k}. \]

### 3.2.2. Estimation of Step-Size

The local curvature estimates \( \theta_k, \gamma_k \) can often be estimated simply from the results of iteration \( k \) and an earlier iteration \( k_0 \). The initial value of the spectral step-size can be calculated by using the local curvature estimation, and the ADMM can modify the spectral step-size by updating the dual variables in the iterative process so as to achieve the best penalty parameter and relaxation parameter.

Define

\[ \Delta \lambda_k = \lambda_k - \sigma \lambda_{k_0}, \]
\[ \Delta f_k = z_k - z_{k_0}, \]

where \( \sigma \) is a scaling parameter.

When solving an unconstrained optimization problem, dual variables \( \lambda_k \) and spectral step-size \( \hat{\theta}_k = \frac{1}{\theta_k}, \hat{\gamma}_k = \frac{1}{\gamma_k} \) affect the convergence performance of the MS-RADMM. At present, line search is commonly used to select \( \hat{\theta}_k, \hat{\gamma}_k \). We can overcome the oscillation
phenomenon by adopting a non-monotonic technique. However, when the initial value is taken near a local valley of the function, it is easy to obtain a local extreme value.

To avoid being trapped in a local optimum, a non-monotonic Wolfe-shaped line search strategy is incorporated into the memory gradient method [40]. By combining the iteration information of current and past multiple points, the global convergence of the algorithm is improved.

The dual variable update rule is derived from

\[
\lambda_k = \begin{cases} 
-\lambda_k, & \text{if } k = 1 \\
-(1-\xi_k)\lambda_k + \xi_k\lambda_k, & \text{if } k \geq 2
\end{cases},
\]

(18)

\[
\xi_k = \frac{\zeta \|\lambda_k\|^2}{\|\lambda_k\|^2 + \|\lambda_{k-1}\|^2}, \quad \zeta \in (0, 1).
\]

(19)

Combined with the idea of the Barzilai–Borwein gradient method, the spectral step-size \(\hat{\theta}_k\) is readily obtained as [41]

\[
\hat{\theta}_k = \begin{cases} 
\hat{\theta}_{MG}, & \text{if } 2\hat{\theta}_{MG} > \hat{\theta}_{SD}\hat{\theta}_k \\
\hat{\theta}_{SD} - \hat{\theta}_{MG}, & \text{otherwise}
\end{cases},
\]

(20)

\[
\hat{\theta}_{SD} = \frac{\langle \Delta f_k, \Delta \lambda_k \rangle}{\|\Delta f_k\| \|\Delta \lambda_k\|},
\]

(21)

\[
\hat{\theta}_{MG} = \frac{\langle \Delta f_k, \Delta \lambda_k \rangle}{\|\Delta f_k\|^2}.
\]

(22)

The spectral step-size \(\hat{\gamma}_k\) is solved likewise.

### 3.2.3. Parameter Update Rules

In the case where the linear model assumptions break down or an unstable step-size is produced, we can employ a correlation criterion to verify the local linear model assumptions.

Define

\[
\theta_{cor}^k = \frac{\langle \Delta f_k, \Delta \lambda_k \rangle}{\|\Delta f_k\| \|\Delta \lambda_k\|},
\]

(23)

\[
\gamma_{cor}^k = \frac{\langle \Delta g_k, \Delta \lambda_k \rangle}{\|\Delta g_k\| \|\Delta \lambda_k\|},
\]

(24)

where \(\theta_{cor}^k\) are the correlations between \(\Delta f_k\), and \(\gamma_{cor}^k\) are the correlations between \(\Delta \lambda_k\). The update rules of penalty and relaxation parameters are given by

\[
\rho_{k+1} = \begin{cases} 
\sqrt{\theta_{cor}^k}, & \text{if } \theta_{cor}^k > \epsilon_{cor} \text{ and } \gamma_{cor}^k > \epsilon_{cor} \\
\theta_{cor}^k, & \text{if } \theta_{cor}^k > \epsilon_{cor} \text{ and } \gamma_{cor}^k \leq \epsilon_{cor} \\
\gamma_{cor}^k, & \text{if } \theta_{cor}^k \leq \epsilon_{cor} \text{ and } \gamma_{cor}^k > \epsilon_{cor} \\
\rho_k, & \text{if } \theta_{cor}^k \leq \epsilon_{cor} \text{ and } \gamma_{cor}^k \leq \epsilon_{cor}
\end{cases},
\]

(25)

\[
\alpha_{k+1} = \begin{cases} 
1 + \frac{2\sqrt{\theta_{cor}^k}}{\theta_{cor}^k + \gamma_{cor}^k}, & \text{if } \theta_{cor}^k > \epsilon_{cor} \text{ and } \gamma_{cor}^k > \epsilon_{cor} \\
1.9, & \text{if } \theta_{cor}^k > \epsilon_{cor} \text{ and } \gamma_{cor}^k \leq \epsilon_{cor} \\
1.1, & \text{if } \theta_{cor}^k \leq \epsilon_{cor} \text{ and } \gamma_{cor}^k > \epsilon_{cor} \\
1.5, & \text{if } \theta_{cor}^k \leq \epsilon_{cor} \text{ and } \gamma_{cor}^k \leq \epsilon_{cor}
\end{cases},
\]

(26)

where \(\epsilon_{cor}\) is the threshold of curvature estimation and is a constant, which is set as 0.2 in this paper with reference to paper [41]. The setting of this threshold further avoids the problem of inaccurate curvature estimation and ensures convergence.
4. RELM Based on the Scalars MS-ARADMM

The MS-ARADMM is employed to solve the convex model-fitting problem of the RELM and improves the convergence speed of the RELM.

4.1. Scalars MS-RADMM for RELM

For an \( m \)-category classification problem, calculation of the RELM objective function (3) is equivalent to (4). First, the hidden layer output matrix \( H \) is acquired by using the RELM. Then, the MS-ARADMM algorithm is used to solve the optimal output weight of the RELM. The iteration process is given by

\[
\begin{align*}
\beta^{k+1}_{im} &= \beta^k_{im} - (H^T_i H_i)^{-1} \frac{\alpha}{L} (a_i^T (H_i^k \beta^k_{im} - z^k_m + \frac{\lambda^k_m}{\alpha \rho})) \\
y^{k+1}_{jm} &= H^T_j \beta^{k+1}_m \\
z^{k+1}_{jm} &= \frac{1}{1 + (\alpha^2 \rho)} y^{k+1}_{jm} + \frac{\alpha^2 \rho}{1 + (\alpha^2 \rho)} (y^{k+1}_{jm} + \frac{\lambda^{k+1}_{jm}}{\alpha \rho}) \\
\lambda^{k+1}_{jm} &= \lambda^{k}_{jm} + \alpha \rho (y^{k+1}_{jm} - z^{k+1}_{jm})
\end{align*}
\]

where \( H_i \) and \( H_j \) are the \( j \)th row and the \( i \)th column of the matrix \( H \), \( k \) represents the number of iterations, and \( m \) represents the number of columns in the matrix. The schematic diagram of the specific model is shown in Figure 1.

![Figure 1. Illustration of the MS-ARADMM-based RELM.](image_url)

4.2. Learning Algorithm for MS-ARADMM-Based RELM

By adding step-size selection constraints in the MS-ARADMM iteration, it is ensured that the penalty and relaxation parameters can converge under the bounded conditions. The steps are shown in Algorithm 1.

**Algorithm 1: MS-ARADMM-based RELM**

| **Data:** the training dataset \( x \), the number of hidden nodes \( L \), regularization parameter \( \mu \), and the number of iterations \( I \) |
| **Result:** the optimal output weight matrix \( \beta \) |

1. for \( i = 1 : I \) |
2. Initialization; |
3. Generation of the hidden-layer output matrix \( H \); |
4. Let the initial values of \( \beta, z, \lambda \); |
5. Perform the MS-ARADMM iterations with (27) and increase \( i \) by 1; |
6. Check the termination condition; |
7. end; |
8. Calculate step-size \( \hat{\theta}_k, \hat{\gamma}_k \) with (20); |
9. Compute the correlation parameters \( \theta^\text{cor}_k, \gamma^\text{cor}_k \) with (23)–(24); |
10. Estimate the best parameters \( \rho, \alpha \) with (25)–(26); |
11. Determine \( \beta, z, \lambda \) under the current iteration number with (27); |
12. Return \( \beta \);
5. Simulation Experiment and Result Analysis

The MS-ARADMM-based RELM is used to train single hidden-layer feedforward neural networks (SLFNs) on eight datasets. The performances of the MS-ARADMM, the MS-RADMM, and the RS-ADMM are evaluated by convergence speed and time cost. The specifications of the datasets are shown in Table 1.

Table 1. Dataset specifications.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Attributes</th>
<th>Number of Training Samples</th>
<th>Number of Testing Samples</th>
<th>Number of Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gisette</td>
<td>5000</td>
<td>5600</td>
<td>1400</td>
<td>2</td>
</tr>
<tr>
<td>USPS</td>
<td>256</td>
<td>7439</td>
<td>1859</td>
<td>10</td>
</tr>
<tr>
<td>Magic</td>
<td>10</td>
<td>15,216</td>
<td>3804</td>
<td>2</td>
</tr>
<tr>
<td>BASEHOCK</td>
<td>4862</td>
<td>1595</td>
<td>438</td>
<td>2</td>
</tr>
<tr>
<td>Pendigits</td>
<td>16</td>
<td>8794</td>
<td>2198</td>
<td>10</td>
</tr>
<tr>
<td>Optical-Digits</td>
<td>64</td>
<td>4496</td>
<td>1124</td>
<td>10</td>
</tr>
<tr>
<td>Statlog</td>
<td>36</td>
<td>5148</td>
<td>1287</td>
<td>6</td>
</tr>
<tr>
<td>PCMAC</td>
<td>3289</td>
<td>1555</td>
<td>388</td>
<td>2</td>
</tr>
</tbody>
</table>

5.1. Performance Analysis of Adaptive Parameter Selection Methods

According to the principle of iterative calculation of the gradient algorithm, since the computational complexity of each iteration is the same, it means that the total number of iterations is positively correlated with the time cost. In other words, the convergence performance of the algorithm can be evaluated by comparing the iterative convergence curves of the algorithm, and the time cost of the algorithm can also be analyzed by analyzing the convergence curves.

In order to verify the convergence of the adaptive parameter selection method, numerical experiments were carried out under the same environmental conditions and compared with the current popular improved Barzilai–Borwein algorithms (MBBH, NABBH, and MTBBH). The effectiveness of the method was evaluated by comparing the total number of iterations at the end of execution.

The MBBH algorithm modifies the standard Barzilai–Borwein step-size [42] to have specific quasi-Newton characteristics. However, the curvature condition is not added, and the generated approximate Hessian matrix cannot meet the iterative requirements, which affects the speed of the algorithm.

The NABBH algorithm improves the convergence speed by simplifying the computational complexity of the inverse operation of Hessian matrix [43]; that is, only the inverse matrix of the first derivative matrix of the function is calculated, and the second derivative matrix of the function is omitted. A step-size selection strategy is designed to speed up the convergence of the algorithm. However, this algorithm fails to converge if the condition of monotonic decrease is not met at each iteration.

The MTBBH algorithm realizes monotonic descent by replacing the exact Hessian with a positive-definite data matrix. However, due to adoption of a non-monotonic technique, the algorithm easily falls into local optima.

For problems that tend to fall into local extremes, a new Barzilai–Borwein-type gradient method is proposed by modifying the original Barzilai–Borwein step-size. By introducing a non-monotonic Wolfe-type strategy into the memory gradient method, the global optimal solution is obtained. The purpose of improving convergence speed is achieved by adding step-size constraints [43]. In theory, the proposed adaptive parameter selection method has better global convergence and convergence speeds.

A comparison of the performances of the MBBH, the NABBH, the MTBBH, and the proposed algorithms through tests was made. Table 2 and Figure 1 show the simulation results of different methods, which demonstrate the correctness of our theoretical analysis. According to Table 2, under different constraint conditions, the proposed method is found to terminate with the least number of iterations, indicating that it has the fastest convergence.
speed. It is also clearly shown in Figure 2 that the proposed method has better global convergence and non-monotonicity than the other algorithms.

Table 2. Comparison of iterations for different methods.

<table>
<thead>
<tr>
<th>Termination Condition (Error)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MBBH</td>
</tr>
<tr>
<td>$1 \times 10^{-4}$</td>
<td>134</td>
</tr>
<tr>
<td>$1 \times 10^{-6}$</td>
<td>182</td>
</tr>
<tr>
<td>$1 \times 10^{-8}$</td>
<td>201</td>
</tr>
<tr>
<td>$1 \times 10^{-10}$</td>
<td>274</td>
</tr>
<tr>
<td>$1 \times 10^{-12}$</td>
<td>306</td>
</tr>
</tbody>
</table>

Figure 2. Convergence comparison of different methods.

5.2. Convergence Analysis

The key performances of the classification model is convergence speed and accuracy. Considering the background of big data, this paper focuses on the convergence speed of the model in algorithm optimization. In order to evaluate the effectiveness of the proposed algorithm, the convergence of the proposed algorithm is evaluated by comparing the time cost, the number of iterations convergence, and the classification accuracy with the newer improved ADMM algorithm.

The proposed MS-ARADMM is compared with the MS-AADMM and RB-ADMM methods on eight datasets. The experiment is conducted by setting the same termination conditions. The evaluation indicators include the number of iterations and computational time.

The RB-ADMM algorithm decomposes the objective function of the model into a loss function and a regularization function; it uses the ADMM to transform the least square problem into the least square problem without a regularization term so as to improve the calculation speed of the model. However, this method does not fully utilize the model structure of the ADMM, leading to slow convergence.

The MS-AADMM adopts a tunable step-size to accelerate convergence. However, parameter selection plays an important role in the convergence of the algorithm. Inappropriate parameter selection may cause the algorithm to not converge.
The MS-ARADMM is realized by employing an adaptive parameter selection method to improve the convergence speed.

Given the hidden-layer output matrix $H$, the optimal output weights of the MS-ARADMM are calculated with (27). The output weights of the RB-ADMM and the MS-AADMM are updated by the following:

$$
\beta_{k+1} = \arg \min_{\beta_k} \left\{ g(\beta_k) + \frac{\rho}{2} \left\| H\beta_k - z_k + \frac{\lambda_k}{\rho} \right\|^2 \right\},
$$

$$
z_{k+1} = \arg \min_{z_k} \left\{ H\beta_k - z_k + \frac{\rho}{2} \left\| H\beta_k - z_k + \frac{\lambda_k}{\rho} \right\|^2 \right\},
$$

$$
\lambda_{k+1} = \lambda_k + \rho (H\beta_{k+1} - z_{k+1})
$$

5.2.1. Comparison of Convergence of MS-ARADMM and RB-ADMM

The difference between the output weight updates (27) and (28) is that all iterations in the MS-ARADMM are for scalar variable updates. The update method of scalar variables simplifies the sub-problem solving, thus improving the convergence. Although the RB-ADMM can adaptively choose penalty parameters to improve the convergence to a certain extent, it suffers from several flaws. The performance of the RB-ADMM can vary wildly depending on the problem size. Furthermore, without a suitable choice of a residual balancing factor, the algorithm may not converge. Aiming at solving the problems of the RB-ADMM, the MS-ARADMM implements adaptive parameter selection by adding step-size selection constraints, thereby improving the convergence.

The simulation results are given in Table 3. As can be seen from Table 3, with the optimization of the algorithm, the time and number of iterations spent by the model to process large-scale data become less and less, which also means that the algorithm proposed in this paper has a better convergence speed. At the same time, to see the improvement effect of the MS-ARADMM algorithm, the algorithm improvement effect calculated from the results in Table 3 is given in Table 4. This can be seen from Table 4 for the convergence speed improvement, in which the convergence speed of the MS-ARADMM is increased by an average of 99.3032% compared with the RB-ADMM in the two-category datasets. In the six-category datasets, compared with the RB-ADMM, the convergence speed of the MS-ARADMM is increased by 98.4375% on average. In the ten-category classification datasets, from Table 4, the convergence speed of the MS-ARADMM is increased by an average of 96.7624% compared with the RB-ADMM.

Table 3. Comparison of the convergence speed of RELM for different algorithms.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RB-ADMM Time(s)</th>
<th>RB-ADMM Number of Iterations</th>
<th>MS-AADMM Time(s)</th>
<th>MS-AADMM Number of Iterations</th>
<th>MS-ARADMM Time(s)</th>
<th>MS-ARADMM Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gisette</td>
<td>737.5244</td>
<td>912</td>
<td>33.4513</td>
<td>39</td>
<td>5.9518</td>
<td>7</td>
</tr>
<tr>
<td>USPS</td>
<td>2621.8914</td>
<td>546</td>
<td>152.5741</td>
<td>31</td>
<td>33.9495</td>
<td>5</td>
</tr>
<tr>
<td>BASEHOCK</td>
<td>332.8139</td>
<td>1674</td>
<td>9.0221</td>
<td>41</td>
<td>1.5389</td>
<td>7</td>
</tr>
<tr>
<td>Magic</td>
<td>1156.6178</td>
<td>674</td>
<td>46.0599</td>
<td>25</td>
<td>12.9115</td>
<td>7</td>
</tr>
<tr>
<td>Pendigits</td>
<td>3912.3798</td>
<td>742</td>
<td>176.3753</td>
<td>33</td>
<td>37.3298</td>
<td>17</td>
</tr>
<tr>
<td>Optical-Digits</td>
<td>1825.9015</td>
<td>538</td>
<td>139.2235</td>
<td>41</td>
<td>24.5017</td>
<td>35</td>
</tr>
<tr>
<td>Statlog</td>
<td>1626.4322</td>
<td>704</td>
<td>92.0177</td>
<td>39</td>
<td>16.4330</td>
<td>11</td>
</tr>
<tr>
<td>PCMAC</td>
<td>353.2414</td>
<td>1753</td>
<td>3.3102</td>
<td>15</td>
<td>1.9676</td>
<td>9</td>
</tr>
</tbody>
</table>
Table 4. Comparison of convergence speed improvement of MS-ARADMM and different methods.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Categories</th>
<th>MS-ARADMM Compared to RB-ADMM</th>
<th>MS-ARADMM Compared to MS-AADMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gisette</td>
<td>2</td>
<td>99.2324</td>
<td>82.0512</td>
</tr>
<tr>
<td>Magic</td>
<td>2</td>
<td>99.5818</td>
<td>82.9268</td>
</tr>
<tr>
<td>BASEHOCK</td>
<td>2</td>
<td>99.4865</td>
<td>72</td>
</tr>
<tr>
<td>PCMAC</td>
<td>2</td>
<td>98.9614</td>
<td>40</td>
</tr>
<tr>
<td>statlog</td>
<td>6</td>
<td>98.4375</td>
<td>71.7948</td>
</tr>
<tr>
<td>USPS</td>
<td>10</td>
<td>99.0842</td>
<td>83.8709</td>
</tr>
<tr>
<td>Pendigits</td>
<td>10</td>
<td>97.7088</td>
<td>48.4848</td>
</tr>
<tr>
<td>Optical-Digits</td>
<td>10</td>
<td>93.4944</td>
<td>14.6341</td>
</tr>
</tbody>
</table>

5.2.2. Comparison of Convergence of MS-ARADMM and MS-AADMM

As with the calculation formula of $\beta$ in MS-ARADMM, the introduction of the scalar variable update method in MS-AADMM leads to much more efficient computation. However, parameter selection must be addressed. From the MS-AADMM perspective, this manner does not take into account that relaxation techniques can further accelerate the convergence. MS-ARADMM simplifies the calculation by designing an adaptive parameter selection method to jointly adjust the penalty and relaxation parameters.

From Table 3, the convergence speed becomes faster and faster. This can also be seen from Table 4 for the convergence speed improvement; the convergence speed of the MS-ARADMM is increased by an average of 69.2445% compared to the MS-AADMM in the two-category datasets. In the six-category datasets, compared to the MS-AADMM, the convergence speed of the MS-ARADMM is increased by 71.7948% on average. In the ten-category classification datasets, from Table 3, the convergence speed of the MS-ARADMM by an average of 48.9966% compared to the MS-AADMM.

For the case of the PCMAC, Pendigits, or Optical-Digits dataset, due to the limited dimension and size of the dataset, this dataset leads to lower improvements in the convergence speed. For instance, in the ten-category datasets, the USPS dataset already achieves improvements of 83.8709%. However, the Optical-Digits dataset only achieved improvements of 14.6341%. This huge difference arises from the fact that the MS-ARADMM is suitable for large-scale optimization problems. This greatly reduces the convergence speed improvement for the Optical-Digits dataset, because the size of the Optical-Digits dataset is $64 \times 5620$ and that of the USPS dataset is $256 \times 9298$.

5.2.3. Convergence Rate Comparison

Implicit in the MS-ARADMM is the assumption of automatically tuning the parameters to achieve an optimal performance. On this basis, we show that the MS-ARADMM generally gives better convergence than other algorithms.

The convergence performance of different algorithms is compared on eight benchmark datasets. Table 3 and Figure 3 show the simulation results of the three algorithms. The results are in full agreement with the theoretical analysis. According to Table 3, the MS-ARADMM algorithm has the lowest computational complexity and the least iterations of all datasets among all algorithms. From Figure 3, with a maximum of 2000 iterations and an error of $10^{-3}$, the MS-ARADMM can meet the termination condition within the minimum number of iterations.

5.3. Parallelism Analysis

Parallelism is an important indicator for evaluating the convergence speed of the ADMM algorithms. High parallelism performance can effectively relieve the computational burden and improve algorithm efficiency. To verify that the MA-ARADMM has a better convergence speed, simulations are carried out on the datasets. The parallelism performance of the MS-ARADMM is evaluated by analyzing the GPU acceleration ratios and the relationship between the acceleration ratios and the number of CPU cores.
5.3.1. Parallel Implementation on Multicore Computers

Using a maximally splitting technique, the RLS problem can be maximally split into univariate sub-problems that can be executed in parallel, leading to a highly parallel structure.

To verify our theoretical analysis, experiments are conducted on the Gisette dataset on different multicore computers. The relationship between acceleration ratios and the number of cores is characterized by the acceleration ratio $R$, defined by the single-core runtime divided by the $n$-core runtime. The experiments are carried out on three multi-core computers. The hardware configurations of the three computers are, respectively, an Intel Core i7-10700 8-core CPU @ 2.9 GHz, an Intel Core i7-4790 4-core CPU @ 3.60 GHz, and an Intel Core i7-8700 6-core CPU @ 3.2 GHz.

The three computers are shown in Figure 4. From Figure 4, the relationship between the acceleration ratios and the number of CPU cores is close to the lower bound, demonstrating the high parallelism of the MS-ARADMM.

Figure 4. Convergence of MS-ARADMM on three computers on Gisette dataset.
5.3.2. Parallel Implementation on GPU

As one of the important indexes to evaluate the convergence performance of the algorithm, the high parallel performance effectively alleviates the computational pressure and further improves the operation efficiency of the algorithm. Through internal multi-process parallel computing, the GPU can have a speed that is one order of magnitude higher than the CPU; it also has a strong ability of floating point arithmetics, which can greatly improve the computing speed of the ADMM and shorten the calculation time.

In case of high dimensional data, MI-based RELM requires a large amount of storage and computation. To verify the high parallelism of the algorithm, parallel accelerated experiments of MS-ARADMM-based and MI-based RELMs are realized on an NVIDIA GeForce GT 730 display card. The parallel implementations on the GPU are implemented by using the gpuArray function in the MATLAB toolbox.

The MS-ARADMM-based RELM splits the model-fitting problem into a set of univariate sub-problems that can be executed in parallel. Its convergence speed is improved by the parameter selection scheme. Theoretically, the MS-ARADMM has good convergence speed and parallelism.

The simulation results from all of the datasets are given in Table 5. From Table 5, on all the datasets except USPS, Pendigits, and Optical-Digits, the computational complexity of the MS-ARADMM-based RELM is much smaller than that of the MI-based RELM. On all of the datasets, the computational complexity of the MS-ARADMM-based RELM is much smaller than that of the MI-based RELM when implemented on the GPU. The acceleration ratio of the MI-based method is about 5.3443, whereas that of the MS-ARADMM is about 23.5065—an acceleration of four times that of the MI-based method.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MI-Based RELM CPU</th>
<th>MI-Based RELM GPU</th>
<th>MI-Based RELM Acceleration Ratio</th>
<th>MS-ARADMM-Based RELM CPU</th>
<th>MS-ARADMM-Based RELM GPU</th>
<th>MS-ARADMM-Based RELM Acceleration Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gisette</td>
<td>51.5313</td>
<td>9.885</td>
<td>5.2131</td>
<td>6.3520</td>
<td>0.715</td>
<td>8.8839</td>
</tr>
<tr>
<td>USPS</td>
<td>50.1406</td>
<td>10.487</td>
<td>4.7812</td>
<td>61.1283</td>
<td>2.589</td>
<td>23.6108</td>
</tr>
<tr>
<td>BASEHOCK</td>
<td>29.9844</td>
<td>5.544</td>
<td>5.4084</td>
<td>1.2750</td>
<td>0.107</td>
<td>11.9159</td>
</tr>
<tr>
<td>Magic</td>
<td>78.7500</td>
<td>13.093</td>
<td>6.0170</td>
<td>29.0134</td>
<td>1.537</td>
<td>18.8766</td>
</tr>
<tr>
<td>Pendigits</td>
<td>54.7188</td>
<td>9.226</td>
<td>5.9309</td>
<td>115.0015</td>
<td>3.043</td>
<td>37.7921</td>
</tr>
<tr>
<td>Optical-Digits</td>
<td>36.2344</td>
<td>6.559</td>
<td>5.5244</td>
<td>44.7947</td>
<td>1.881</td>
<td>23.8143</td>
</tr>
<tr>
<td>Statlog</td>
<td>38.5156</td>
<td>8.756</td>
<td>4.3988</td>
<td>34.5882</td>
<td>0.675</td>
<td>51.2418</td>
</tr>
<tr>
<td>PCMAC</td>
<td>27.9688</td>
<td>5.103</td>
<td>5.4809</td>
<td>2.7872</td>
<td>0.230</td>
<td>12.1183</td>
</tr>
</tbody>
</table>

5.4. Accuracy Analysis

The classification accuracy is an important indicator of classifier performance. Accuracy was compared on the MS-ARADMM-based, MS-AADMM-based, and MI-based RELMs. Table 6 compares the training accuracy and the testing accuracy of the MS-ARADMM, the MS-AADMM, and the MI-based RELMs. From Table 6, we can see that the classification accuracy by the MS-ARADMM is not affected. From Tables 3 and 6, under approximately identical training and the testing accuracy, the computational time for the MS-ARADMM is less than those of both the MI-based and the MS-AADMM-based methods. Thus, the convergence speed of MS-ARADMM is greatly improved in solving large-scale optimization problems.
Table 6. Training accuracy and testing accuracy.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MI-Based Training</th>
<th>MI-Based Testing</th>
<th>MS-AADMM Training</th>
<th>MS-AADMM Testing</th>
<th>MS-ARADMM Training</th>
<th>MS-ARADMM Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gisette</td>
<td>99.1607</td>
<td>95.5714</td>
<td>99.6071</td>
<td>96.2143</td>
<td>99.1607</td>
<td>96.0714</td>
</tr>
<tr>
<td>USPS</td>
<td>97.9161</td>
<td>97.0430</td>
<td>98.0774</td>
<td>96.5591</td>
<td>98.0371</td>
<td>96.9054</td>
</tr>
<tr>
<td>BASEHOCK</td>
<td>90.5867</td>
<td>90.2133</td>
<td>89.5859</td>
<td>90.1303</td>
<td>91.0915</td>
<td>90.4111</td>
</tr>
<tr>
<td>Magic</td>
<td>81.9203</td>
<td>81.6812</td>
<td>81.7889</td>
<td>81.7560</td>
<td>81.7737</td>
<td>81.7297</td>
</tr>
<tr>
<td>Pendigits</td>
<td>96.7023</td>
<td>96.6788</td>
<td>96.5658</td>
<td>96.6968</td>
<td>96.5203</td>
<td>96.8148</td>
</tr>
<tr>
<td>Optical-Digits</td>
<td>98.8657</td>
<td>98.3986</td>
<td>98.7989</td>
<td>98.8486</td>
<td>98.8434</td>
<td>97.9537</td>
</tr>
<tr>
<td>Statlog</td>
<td>85.8197</td>
<td>82.5174</td>
<td>85.6255</td>
<td>82.7506</td>
<td>85.7031</td>
<td>82.5951</td>
</tr>
<tr>
<td>PCMAC</td>
<td>99.4852</td>
<td>91.5167</td>
<td>99.3565</td>
<td>91.5167</td>
<td>99.7426</td>
<td>91.4602</td>
</tr>
</tbody>
</table>

6. Conclusions

In this paper, an MS-ARADMM algorithm is proposed to solve the RLS problem in the RELM. Its novelty is reflected in two aspects: (1) The non-monotonic Wolfe-type strategy is introduced into the memory gradient method to improve the global convergence; (2) The step selection constraint is added to simplify the computational complexity of MS-RADMM subproblems. Since the MS-ARADMM is a convex optimization method with superlinear global convergence, it can ensure a fast response and global optimal solution of the RELM, so it is more suitable to realize the distributed computation of large-scale convex optimization problems of the RELM compared with other ADMM methods.

We focused on the influences of parameters $\rho$ and $\alpha$ on the convergence performance of the RELM model. To verify the performance of the proposed algorithms, we applied them to various large-scale classification datasets, and compared the simulation results with the methods implemented in Tables 2 and 3. The results confirm that the computation efficiency of the RELM model is obviously improved, especially when applied to large-scale convex optimization problems. Therefore, the MS-ARADMM algorithm could enhance the convergence speed since it has a simpler solution process.

Author Contributions: Conceptualization, S.H. and Z.W.; methodology, Z.W.; writing—original draft preparation, S.H. and Z.W.; writing—review and editing, X.X., B.L. and K.W.; supervision, Z.W. and B.L.; project administration, Z.W. and K.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Zhejiang Provincial “Ling Yan” Research and Development Program under Grant No. 2022C03122, in part by Public Welfare Technology Application and Research Program in Zhejiang Province under Grants No. LGF22F020006 and LQ23F030002, and in part by the Open Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University under No. ICT2022B34.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.