Algorithms for Accelerating Machine Learning with Wide and Deep Models

Wide・Deepモデルを用いた機械学習を
高速化するためのアルゴリズム

Yasutoshi Ida

井田 安俊

Abstract

Advances in information technologies have made it possible to store a wide variety of large-scale data. However, data must be utilized, not just stored, to create value for society. Machine learning is one of the most promising methods to utilize such data. We are already surrounded by machine learning-based technologies. Machine learning has become a part of our life, and the development of advanced machine learning technologies will enrich our lives.

Machine learning can be broadly defined as computational methods that increase the performance of a particular task by learning certain models from data. In the era of big data, large-scale data is expected: however, in recent years, the size of models is also increasing. This is because increasing the size of models enables machine learning to improve the accuracy of certain tasks and handle large data. Such large models are already used in the field of bioinformatics, computer vision, and natural language processing.

In this dissertation, we propose fast algorithms to reduce the processing times for large models. The additional capital investment cost is not required by utilizing algorithmic solutions. This dissertation focuses on wide and deep models in large models that perform an important role in machine learning, i.e., wide models are used for analyzing high-dimensional data such as genome data and deep models are crucial for recent applications in artificial intelligence.

For wide models, we primarily handle feature selection on the basis of linear regression models with high-dimensional data. The key to the proposed algorithms is to utilize sparsity in the models. Our algorithms safely skip unnecessary computations and intensively learn important parts in the model. In addition, our approach does not require additional hyperparameters that incur additional tuning costs. Experiments verify that our method reduces the processing times of wide models without

degrading accuracy.

While considering deep models, we handle deep neural networks (DNNs), which are fundamental models in recent artificial intelligence technologies. As DNNs require long processing times for training (learning phase) and inference (prediction phase), which is a crucial problem, we propose two efficient algorithms, one for training and the other for inference. For training, we propose an optimization method with preconditioning to effectively reduce the training loss. For inference, we propose a model compression method that erases unimportant parts from deep models and produces lightweight models. Experiments demonstrate that our algorithms can effectively reduce the training loss or model size while maintaining the accuracy.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Hisashi Kashima at Kyoto University, for his continuous guidance and valuable advice. Moreover, I would like to express my sincere thanks to the committee professors of my thesis, Prof. Toshiyuki Tanaka and Prof. Nobuo Yamashita, for their constructive comments and discussions.

I would like to express my gratitude to Dr. Yasuhiro Fujiwara who taught me how to conduct my research at Nippon Telegraph and Telephone Corporation (NTT). I would also like to thank all the researchers who have collaborated with me, including Dr. Sotetsu Iwamura, Dr. Tomoharu Iwata, Dr. Koh Takeuchi, and Dr. Sekitoshi Kanai. Our discussions were significantly useful for my research. I would like to express my gratitude to Dr. Takashi Matsumoto and Tomohiko Suzuki. They taught me the basics of machine learning and inspired me to become a researcher.

Finally, I would like to thank my family for their continuous support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Machine Learning

Machine learning can be broadly defined as computational methods that increase the performance of some task by learning some model from data (Mitchell, 1997; Shalev-Shwartz and Ben-David, 2014; Mohri et al., 2018). It has become one of the most crucial information technologies as the amount of data available increases. We are already surrounded by machine learning-based technology as follows:

- A number of personal computers and smart speakers install intelligent personal assistants that perform tasks for humans on the basis of their voice or text commands: they use machine learning to recognize these commands.

- Facial recognition systems on smartphones learn from face data to perform biometric authentication for unlocking a device and making payments.

- Cars with a driving safety support system prevent traffic accidents by utilizing machine learning-based systems such as traffic sign recognition and lane-keeping assist.

We can also search for various contents such as documents, images, movies and songs thanks to search engines using machine learning, and satellite navigation software learns from traffic data to estimate travel times. It is also widely used in scientific applications such as bioinformatics, medicine, and astronomy. Machine learning

has become a part of our life, and the development of advanced machine learning technologies will enrich our lives.

1.2 Wide and Deep Models

Models in machine learning are learned from collected data with learning algorithms. With the increase of the size of collectable data thanks to the progress of information technologies, the size of models used in machine learning has also increased. Although various types of large models exist, this dissertation focuses on wide and deep models. The wide model means that its parameters to be learned take the form of a long vector or a wide matrix as shown in Figure 1.1. On the other hand, the deep model has a deep hierarchical structure in the parameters as described in Figure 1.2. The following examples are famous wide and deep models which are relevant to this dissertation:

Lasso. Lasso (Tibshirani, 1996) is a popular feature selection method for **wide models** in machine learning (Li et al., 2017). Since it can find some important features for some tasks from high-dimensional data, it has been used for a long time to analyze data. In genome data analysis, lasso is often used to select SNP (single nucleotide polymorphism) related to disease from human genome (Wu et al., 2009): the dimensions of the data reach millions and more (1000 Genomes Project Consortium and others, 2015). In such a case, the width of the model also reaches more than millions. Since there are many high-dimensional data in the real world such as text data, network traffic data, bibliographic data, and hyperspectral medical images, feature selection is a fundamental tool for analyzing such data.

Deep learning. Deep learning (Goodfellow et al., 2016) is a subfield of machine learning that uses **deep models**. They have hierarchical structures and the depth of the hierarchy can reach more than 100 (He et al., 2016b). Thanks to the deep hierarchical structure, deep learning has achieved high accuracy for various tasks such as image recognition (Krizhevsky et al., 2012), speech recognition (Seide et al., 2011), and machine translation (Sutskever et al., 2014). For this reason, deep learning is widely used in various fields such as com-

puter vision, speech processing, and natural language processing, and a crucial fundamental technology for artificial intelligence.

As shown in the examples above, the wide and deep models play important roles across various research fields.

1.3 Computation Cost

The previous section describes that large models in machine learning have contributed to society via contributions to various research fields. However, the computation cost has clearly increased due to the large size of the models.

In the case of lasso, it incurs high computation cost as the width becomes larger. For instance of graph construction (Meinshausen and Buhlmann, 2006), the width of lasso corresponds to the number of data points. For such a situation, lasso cannot finish the computation within a month when we use more than one million data points (Fujiwara et al., 2016a).

Deep learning obviously requires a long processing time due to the deep hierarchical structure. In the field of computer vision, AlexNet, a model used in deep learning, won an ILSVRC image classification competition with 8 layers in 2012 (Krizhevsky et al., 2012). In the same competition in 2015, the deep model called Residual Network (ResNet) used 152 layers and won the competition (He et al., 2016b). Due to the increase in the number of layers, ResNet requires 20 times longer processing time than AlexNet for the inference (Bianco et al., 2018).

As we described above, wide and deep models are computationally expensive although they are fundamental models for machine learning applications. We need efficient methods for these models to make machine learning more accessible and easy to use.

1.4 Fast Methods for Wide and Deep Models

In this dissertation, we develop fast algorithms for large models to reduce the processing time. Although various types of large models exist in machine learning, we focus on the wide and deep models as we have described in the previous sections.

This dissertation is divided into two parts. In the first part, we develop fast algorithms for typical wide models whose parameters take the form of a long vector (Figure 1.1 (a), Chapter 2) and a large matrix (Figure 1.1 (b), Chapter 3). In the second part, we focus on deep neural networks: they are typical deep models that have hierarchical structures for parameters (Figure 1.2). We propose algorithms to speed up the training (Chapter 4) and the inference (Chapter 5).

We list the contributions to be presented in each chapter of the dissertation:

Part I: Fast algorithms for wide models.

Chapter 2: Linear regression model for high-dimansional feature vector is a typical wide model. Since its parameters take the form of a long parameter vector corresponding to a high-dimensional feature vector, it requires a long processing time for learning. In this chapter, we focus on linear regression models with structured sparsity-inducing norms (Jenatton, 2011). By introducing the norms to the models, a lot of elements in the parameter vector turn to be zeros. We propose a fast algorithm without degrading accuracy on the basis of the property: it safely skips unnecessary computations by utilizing the sparsity. Another advantage of structured sparsity-inducing norms is the ability to handle a wide variety of structured data. Although we start with Sparse Group Lasso (Simon et al., 2013) that can handle data with group structures, we show that our algorithm can be extended to data with overlapping groups (Overlapping Group Lasso) and graph structures (Graph Lasso) (Jacob et al., 2009).

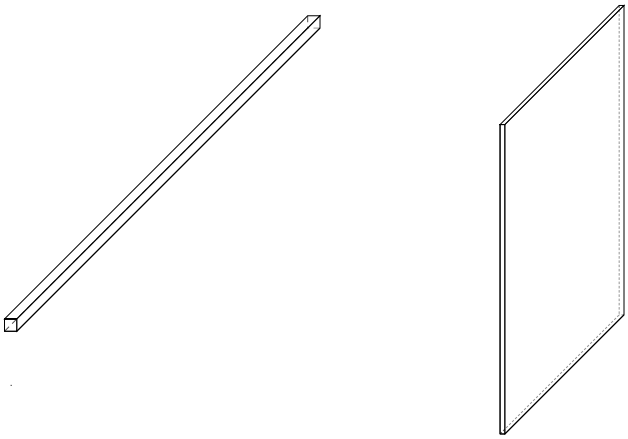
Chapter 3: The deterministic CUR matrix decomposition (Bien et al., 2010) is a low-rank approximation method for a data matrix on the basis of a regularized self-representation model (Zhu et al., 2015). Its parameter takes the form of a matrix, and the size of the parameter matrix increases with the size of the data matrix: CUR matrix decomposition requires a long processing time when it decomposes a large data matrix. In this chapter, we propose a fast algorithm for the deterministic CUR matrix decomposition by utilizing the sparsity in a regularized self-representation model similarly to Chapter 2. Experiments show that our method is up to $10\times$ faster than the original method, and up to $4\times$ faster than the state-

of-the-art method while achieving the same accuracy without additional hyperparameters.

Part II: Fast algorithms for deep models.

Chapter 4: Deep Neural Networks (DNNs) are major models in the field of recent artificial intelligence including machine learning. They have achieved high accuracy for a wide range of tasks thanks to the hierarchical structures of the parameters. However, the large number of parameters incurs a large computation cost for the training of DNNs. Adaptive learning rate algorithms (Tieleman and Hinton, 2012; Zeiler, 2012; Kingma and Ba, 2014) are used to accelerate the training by adjusting the amount of the update for each parameter in DNNs. This chapter proposes an adaptive learning rate algorithm: the key idea is to effectively handle the noise of the gradient. Experiments demonstrate that our method effectively reduces the training loss especially in the case of a noisy setting.

Chapter 5: This chapter proposes a method accelerating the inference of DNNs while Chapters 4 focuses on accelerating the training. Although DNNs effectively extract features from input data by stacking a lot of layers, the stacking is clearly a cause of the increase of the processing time for the inference. To address this problem, our method erases multiple layers from DNNs without degrading accuracy. Our key idea is to use Residual Networks as the models of DNNs, and introduce a priority term that identifies the importance of layers; we can select unimportant layers according to the priority and erase them after the training. In addition, we retrain the networks to avoid critical drops in accuracy after layer erasure. Our experiments show that our method reduces the number of layers by 24.00 %$\sim$42.86 % without any drop in accuracy and speeds up the inference.



(a) Chapter 1: long vector (b) Chapter 2: large matrix

Figure 1.1: Part I: wide models.

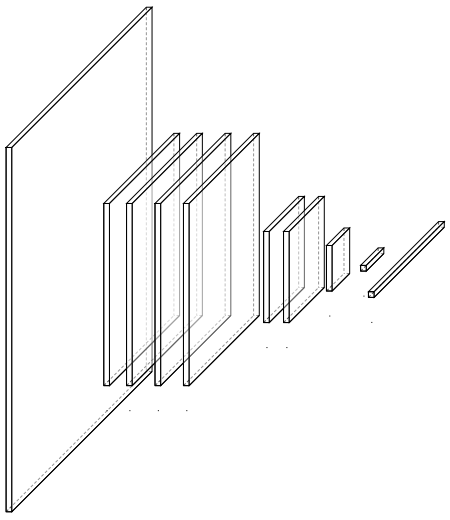


Figure 1.2: Part II: deep models.

Part I

Fast Algorithms for Wide Models

Chapter 2

Fast Block Coordinate Descent for Linear Regression Models with Structured Sparsity-Inducing Norms

2.1 Introduction

Sparse Group Lasso (SGL) (Friedman et al., 2010; Simon et al., 2013) is a popular feature-selection method based on the linear regression model for data that have group structures. For the analysis of such data, it is important to identify not only individual features but also groups of features that have some relationships with the response. SGL finds such groups and features by obtaining sparse parameters corresponding to the features in the linear regression model. In particular, SGL effectively achieves parameter sparsity by utilizing two types of regularizations: feature- and group-level regularization. Owing to its effectiveness, SGL is used in the analysis of various data, e.g., gene expression data (Nam and Kim, 2008; Roth and Fischer, 2008) and climate data (Ndiaye et al., 2016).

In order to obtain the sparse parameters in SGL, Block Coordinate Descent (BCD) is used as a standard approach (Friedman et al., 2010; Simon et al., 2013).

BCD iteratively updates the parameters for each group until convergence. In particular, in each iteration, it first checks whether the parameters in a group are updated to zeros by using all the parameters or data points. If the parameters in the group are determined as nonzeros, BCD updates the parameters in the group. It applies the aforementioned steps to the parameters of each group until the parameters of all groups converge.

Although SGL is practical for analyzing group structured data, BCD suffers from high computation costs. The main bottleneck is the computation to check whether the parameters of a group are updated to zeros, because the computation uses all the parameters or data points. The screening technique is the main existing approach for reducing the computation cost of BCD by reducing the data size (Wang and Ye, 2014; Ndiaye et al., 2015; 2016; 2017). This technique eliminates features and groups whose parameters are zeros, before entering the iterations of BCD. However, the screening techniques cannot be expected to reduce the data size when the initial parameters are far from optimal (Johnson and Guestrin, 2016). The screening techniques often face such problems in practice, and the efficiency of BCD would not be increased in such cases. Therefore, speeding up BCD is still an important topic of study for handling large data.

This chapter proposes a fast BCD for SGL. Our main idea is to identify the groups whose parameters *must be updated to zeros* by only using the parameters in the group, whereas the standard method uses all the parameters or data points. As the number of parameters in one group is typically much smaller than the total number of parameters or data points, our method efficiently skips the computation of groups whose parameters must be updated to zeros. Another idea is to extract a *candidate group set*, which contains groups whose parameters *must not be updated to zeros*. As the parameter groups in the set are likely to largely contribute to the prediction (Fujiwara et al., 2016b;a), we can expect BCD to effectively optimize the objective function by preferentially updating the parameter groups in the set. An attractive point of our method is that it does not need any additional hyperparameters, which incur additional tuning costs. In addition, it provably guarantees convergence to the same value of the objective function as the original BCD. Experiments demonstrate that our method enhances the efficiency of BCD while achieving the same prediction error. Although we first consider the case of non-overlapping

groups, we show that our method is relatively easy to be extended for overlapping groups and graphs by using overlap norm (Jacob et al., 2009).

2.2 Preliminary

2.2.1 Sparse Group Lasso

This section defines the SGL as a method of linear regression analysis that finds small sets of groups in addition to features that achieve high prediction accuracy for the response. Let n be the number of data points, where each data point is represented as a p-dimensional feature vector, $y \in \mathbb{R}^n$ be a continuous response, and G be the total number of feature groups. A data matrix $X \in \mathbb{R}^{n \times p}$ is represented as $X = [X^{(1)}, X^{(2)}, ..., X^{(G)}]$, where $X^{(g)} \in \mathbb{R}^{n \times p_g}$ is the block matrix of X corresponding to the g-th feature group with the number of features p_g. Similarly, parameter vector $\beta \in \mathbb{R}^p$ is represented as $\beta = [\beta^{(1)\mathrm{T}}, \beta^{(2)\mathrm{T}}, ..., \beta^{(G)\mathrm{T}}]^{\mathrm{T}}$, where $\beta^{(g)} \in \mathbb{R}^{p_g}$ is the parameter (regression coefficient) vector of group g. Therefore, the linear regression model in SGL is represented as $y = X\beta = X^{(1)}\beta^{(1)} + \cdots + X^{(G)}\beta^{(G)}$. Solution $\hat{\beta}$ is obtained by solving the following problem:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left(\frac{1}{2n} \| y - \sum_{g=1}^{G} X^{(g)} \beta^{(g)} \|_2^2 + (1-\alpha)\lambda \sum_{g=1}^{G} \sqrt{p_g} \| \beta^{(g)} \|_2 + \alpha\lambda \| \beta \|_1 \right), \quad (2.1)$$

where $\alpha \in [0, 1]$ and $\lambda \geq 0$ are regularization constants; α decides the balance of the convex combination of l_1 and l_2 norm penalties and λ controls the degree of sparsity of the solution.

2.2.2 Block Coordinate Descent

BCD is a standard approach used to obtain solution $\hat{\beta}$ of SGL (Simon et al., 2013). It consists of a group-level outer loop and an element-level inner loop. The group-level outer loop checks whether parameter vector $\beta^{(g)}$ for each feature group is a zero vector. If $\beta^{(g)}$ turns to be a nonzero vector, the element-level loop updates each parameter in $\beta^{(g)}$. The process terminates if the whole parameter vector converges.

In the element-level inner loop, BCD updates $\beta^{(g)}$ in group g if the parameter vector of the group is not a zero vector. The updated parameter vector $\beta^{(g)}_{\mathrm{new}}$ is defined

as follows:

$$\beta_{\text{new}}^{(g)} = \left(1 - \frac{t(1-\alpha)\lambda}{\|S(Z^{(g)}, t\alpha\lambda)\|_2}\right)_+ S(Z^{(g)}, t\alpha\lambda). \tag{2.2}$$

In Equation (2.2), $(\cdot)_+ = \max(0, \cdot)$ and $Z^{(g)} = \beta^{(g)} + \frac{t}{n}(X^{(g)\mathrm{T}} r_{(-g)} - X^{(g)\mathrm{T}} X^{(g)} \beta^{(g)})$, where $t \geq 0$ is the step size. $r_{(-g)}$ is the partial residual and is defined as follows:

$$r_{(-g)} = y - \sum_{l \neq g}^{G} X^{(l)} \beta^{(l)}. \tag{2.3}$$

$S(\cdot, \cdot)$ is the coordinate-wise soft-threshold operator; the j-th element is computed as $S(z, \gamma)[j] = \text{sign}(z[j])(|z[j]| - \gamma)_+$. $\beta^{(g)}$ is iteratively updated in the element-level inner loop using Equation (2.2) until convergence. If the parameter vector of the group is determined as a zero vector, Equation (2.2) is skipped. The computation cost of Equation (2.2) is $\mathcal{O}(p_g^2)$ time because $X^{(g)\mathrm{T}} X^{(g)}$ in $Z^{(g)}$ are precomputed before entering the main loop. In addition, $X^{(g)\mathrm{T}} r_{(-g)}$ has already been computed in the group-level outer loop, as described next.

In the group-level outer loop, $\beta^{(g)}$ is the zero vector if the following condition holds:

$$\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 \leq \sqrt{p_g}(1 - \alpha)\lambda. \tag{2.4}$$

In other words, if Equation (2.4) holds, the parameter vector of group g is a zero vector; Equation (2.2) is then skipped, and the parameter vector is not updated. $X^{(g)\mathrm{T}} r_{(-g)}$ in Equation (2.4) is computed using the following equation that consists of only matrix operations:

$$X^{(g)\mathrm{T}} r_{(-g)} = X^{(g)\mathrm{T}} y - X^{(g)\mathrm{T}} X \beta + X^{(g)\mathrm{T}} X^{(g)} \beta^{(g)}. \tag{2.5}$$

In this equation, $X^{(g)\mathrm{T}} y$, $X^{(g)\mathrm{T}} X$, and $X^{(g)\mathrm{T}} X^{(g)}$ are precomputed before entering the loops. The costs of the precomputations have relatively low impacts on the total computational cost because precomputations are performed only once in the total computation, and are easily parallelized. On the other hand, the computation cost of Equation (2.5) is still high because it requires $\mathcal{O}(pp_g + p_g^2)$ time, and it is repeatedly performed until convergence. As a result, we need $\mathcal{O}(pp_g + p_g^2)$ time for Equation (2.4)

at every iteration. We can modify Equation (2.5) to have $\mathcal{O}(np_g)$ time by maintaining the partial residuals of Equation (2.3) as described in (Huang et al., 2012). However, in either case, the computation cost of Equation (2.4) depends on p or n. Therefore, Equation (2.4) incurs a large computation cost for high-dimensional features or a large number of data points.

2.3 Proposed Approach

In this section, we introduce our algorithm, which efficiently obtains the solution of SGL. First, we explain the ideas underlying our algorithm. Next, we introduce several lemmas that are necessary to derive our algorithm. We then describe the algorithm. We show all the proofs in Section 2.8.

2.3.1 Idea

In SGL, obtaining the solution through BCD incurs a high computation cost. This is because (i) Equation (2.4) requires $\mathcal{O}(pp_g + p_g^2)$ or $\mathcal{O}(np_g)$ time, which incurs a large computation cost for large feature vectors or a large number of data points, and (ii) BCD always checks all of the feature groups using Equation (2.4) at every iteration even when most of the groups have zero vectors.

Our main idea is to identify groups whose parameter vectors *must be updated to zero vectors* by approximating Equation (2.4), which checks whether the parameter vector of each group is a zero. In particular, we compute an upper bound of $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ instead of computing the exact value. If the upper bound is lower than $\sqrt{p_g}(1-\alpha)\lambda$, the parameter vector of the group must be updated to a zero vector. As a result, we can safely skip the computation of the group. As the evaluation of our upper bound requires only $\mathcal{O}(p_g)$ time instead of the $\mathcal{O}(pp_g + p_g^2)$ or $\mathcal{O}(np_g)$ time for the original Equation (2.4), we can effectively reduce the computation cost.

Another idea is to extract a *candidate group set*, which contains groups whose parameter vectors *must not be updated to zero vectors*. As the parameters in the set are likely to largely contribute to the prediction (Fujiwara et al., 2016b;a), we can expect BCD to effectively optimize the objective function by preferentially updating the parameters in the set. In addition, our method only requires $\mathcal{O}(G)$ time to construct the set, and thus the computation cost is relatively low.

2.3.2 Upper Bound for Skipping Computations

We introduce an upper bound of $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ in Equation (2.4). To derive a tight upper bound, we introduce reference parameter vectors and partial residuals of Equation (2.3) that are computed before entering the group-level outer loop. To be specific, we can obtain a tight bound by explicitly utilizing the term representing the difference between the reference and current parameter vectors. As many parameter vectors rapidly converge during the iterations, the difference between the reference and current parameter vectors rapidly decreases. First, we define $U^{(g)}$ that is used to identify groups whose parameter vectors must be updated to zero vectors as follows:

Definition 2.1 *Let $\tilde{r}_{(-g)}$ be a partial residual of Equation (2.3) before entering the group-level outer loop. Then, we define $U^{(g)}$ as follows:*

$$U^{(g)} = \|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2 + \Lambda(g, g) + \textstyle\sum_{l=1}^{G} \Lambda(g, l), \tag{2.6}$$

where $\Lambda(g, l) = \|\hat{K}^{(g)}[l]\|_2 \|\beta^{(l)} - \tilde{\beta}^{(l)}\|_2$. The i-th element of $\hat{K}^{(g)}[l] \in \mathbb{R}^{p_g}$ is given as $\|K^{(g,l)}[i, :]\|_2$, that is, the l_2 norm of the i-th row vector in block matrix $K^{(g,l)} \in \mathbb{R}^{p_g \times p_l}$ of $K := X^{\mathrm{T}} X \in \mathbb{R}^{p \times p}$. $\tilde{\beta}^{(g)}$ is a parameter vector before entering the group-level outer loop.

Note that we can precompute $\|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2$ and $\|\hat{K}^{(g)}[\cdot]\|_2$ before entering the group-level outer loop and the main loop, respectively. Next, the following lemma shows that $U^{(g)}$ is an upper bound of $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$:

Lemma 2.1 (Upper bound) *For each $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ of group g, we have $U^{(g)} \geq \|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$.*

Then, the following lemma shows the property of the upper bound corresponding to groups with parameters that must be updated to zeros:

Lemma 2.2 (Groups with zero vectors) *Parameter $\beta^{(g)}$ for group g is updated to a zero vector if $U^{(g)}$ in Definition 2.1 satisfies $U^{(g)} \leq \sqrt{p_g}(1-\alpha)\lambda$ given $\|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2$, $\|\hat{K}^{(g)}[\cdot]\|_2$, and $\tilde{\beta}$.*

Lemma 2.2 indicates that we can identify groups whose parameters must be updated to zeros by using upper bound $U^{(g)}$ instead of $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$. The error bound of $U^{(g)}$ for $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ is described in a later section.

2.3.3 Online Update Scheme of Upper Bound

Although we can identify groups whose parameters must be updated to zeros by using the upper bound $U^{(g)}$, $\mathcal{O}(p + p_g)$ time is still required to compute Equation (2.6) of the upper bound even if we precompute $\|X^{(g)\mathrm{T}}\tilde{r}_{(-g)}\|_2$ and $\|\hat{K}^{(g)}[\cdot]\|_2$. As the standard approach requires $\mathcal{O}(pp_g + p_g^2)$ or $\mathcal{O}(np_g)$ time, the efficiency of our approach would be moderate. This is the motivation behind our use of the online update scheme for the upper bound that further reduces the computation cost. In particular, when a parameter vector of a group is updated, we use the following updating rule for the upper-bound computation:

Definition 2.2 (Online update scheme of upper bound) *If $\beta^{(g)}$ is updated to $\beta^{(g)'}$, we update upper bound $U^{(g)}$ of Equation (2.6) as follows:*

$$U^{(g)'} = U^{(g)} - 2\Lambda(g, g) + 2\|\hat{K}^{(g)}[g]\|_2\|\beta^{(g)'} - \tilde{\beta}^{(g)}\|_2. \tag{2.7}$$

Equation (2.7) clearly holds because we subtract old values of $2\Lambda(g, g)$ from Equation (2.6), and add updated values of $2\|\hat{K}^{(g)}[g]\|_2\|\beta^{(g)'} - \tilde{\beta}^{(g)}\|_2$ to the equation. In terms of the computation cost, we have the following lemma:

Lemma 2.3 (Computation cost for online update scheme of upper bound) *The computation of Equation (2.7) requires $\mathcal{O}(p_g)$ time given precomputed $\|X^{(g)\mathrm{T}}\tilde{r}_{(-g)}\|_2$ and $\|\hat{K}^{(g)}[\cdot]\|_2$ when the parameter vector of group g is updated.*

The above lemma shows that we can update the upper bound in $\mathcal{O}(p_g)$ time. The computation cost is significantly low compared with the computations of Equations (2.4) and (2.6), which require $\mathcal{O}(pp_g + p_g^2)$ (or $\mathcal{O}(np_g)$) and $\mathcal{O}(p + p_g)$ times, respectively. Therefore, we can efficiently identify groups whose parameters must be updated to zeros on the basis of Lemma 2.2 and Definition 2.2.

2.3.4 Candidate Group Set for Selective Updates

In this section, we introduce a method to extract the *candidate group set*, which contains the groups whose parameters must *not* be updated to zeros. We expect BCD to effectively update the parameter vectors by preferentially updating the parameter vectors on the candidate group set. To extract the candidate group set, we utilize a

criterion, which approximates $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ in Equation (2.4). If the criterion for a group is above a threshold, the group is included in the set. We first define $C^{(g)}$, which is used to check whether the group is included in the candidate group set.

Definition 2.3 *We define $C^{(g)}$ as follows:*

$$C^{(g)} = \|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2 - \alpha\lambda\sqrt{p_g/2}, \tag{2.8}$$

where $\tilde{r}_{(-g)}$ is a partial residual of Equation (2.3) before entering the group-level outer loop.

The error bounds of $C^{(g)}$ and $U^{(g)}$ for $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ are shown as follows:

Lemma 2.4 (Error bound) *Let ϵ be an error bound of $C^{(g)}$ for $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ such that $|C^{(g)} - \|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2| \leq \epsilon$. We then have $\epsilon = \Lambda(g,g) + \sum_{l=1}^{G} \Lambda(g,l) + \alpha\lambda\sqrt{p_g/2}$. In addition, we have $|U^{(g)} - \|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2| \leq 2\epsilon$.*

The above lemma suggests that $C^{(g)}$ approximates $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ better than $U^{(g)}$ because the error bound of $C^{(g)}$ is half the size of that of $U^{(g)}$. We extract a set $\mathbb{C}$ with respect to $C^{(g)}$ by using the following definition:

Definition 2.4 $\mathbb{C}$ *is defined as*

$$\mathbb{C} = \{g \in \{1, ..., G\} | C^{(g)} > \sqrt{p_g}(1-\alpha)\lambda\}. \tag{2.9}$$

Set $\mathbb{C}$ has the following property:

Lemma 2.5 (Groups containing nonzero vectors) $\mathbb{C}$ *in Definition 2.4 given $C^{(g)}$ for $g \in \{1, ..., G\}$ contains the groups whose parameters must be updated to nonzeros.*

The above lemma suggests that $\mathbb{C}$ comprises not only the groups whose parameters must be updated to nonzeros but also groups whose parameters can be updated to nonzeros. Thus, we call $\mathbb{C}$ *candidate group set*. In terms of the computation cost, we have the following lemma to extract the candidate group set:

Lemma 2.6 (Computation cost of candidate group set) *Given precomputed $\|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2$, we can extract candidate group set $\mathbb{C}$ in $\mathcal{O}(G)$ time.*

2.3.5 Algorithm

This section describes our algorithm, which utilizes the above-mentioned definitions and lemmas. Algorithm 2.1 gives a full description of our approach, which is based on BCD with the sequential rule (Ghaoui et al., 2012): a standard approach for SGL. The sequential rule is used to tune regularization constant λ with respect to the sequence of $(\lambda_q)_{q=0}^{Q-1}$, where $\lambda_0 > \lambda_1 > ... > \lambda_{Q-1}$: it sequentially optimizes the parameter vector by using $(\lambda_q)_{q=0}^{Q-1}$, and reuses the solution of the previous λ as the initial parameters for the current λ.

Our main idea is to skip groups whose parameters must be updated to zeros during the optimization by utilizing Lemma 2.2. As upper bound $U^{(g)}$ in Lemma 2.2 can be computed with a low computation cost as described in Lemma 2.3, we can efficiently avoid the computation of Equation (2.4), which is the main bottleneck of the standard BCD. In addition, we extract the candidate group set before we start to optimize the parameters for the current λ. The impact of the computation cost is relatively low on the total cost, as shown in Lemma 2.6. We expect BCD to raise the effectiveness by preferentially updating the parameters in the set based on Lemma 2.5.

In Algorithm 2.1, (lines 2–4), we first precompute $\|\hat{K}^{(g)}[l]\|_2$, which is used for computing the upper bounds. In the loop of the sequential rule, we construct the candidate group set (lines 6–10). Although we compute Equation (2.9) in the initial iteration, we reuse the term $\|X^{(g)\mathrm{T}}\tilde{r}_{(-g)}\|_2$ of the previous iteration in the equation for the other iterations. Next, BCD is performed on the parameter vectors of the set (lines 11–19). Then, the algorithm enters the loop of another BCD with upper bounds (lines 20–36). The reference parameter vector is set (line 21), and $\|X^{(g)\mathrm{T}}\tilde{r}_{(-g)}\|_2$ is precomputed, which is also used for the computation of the upper bounds (lines 22 and 23). In the group-level outer loop, upper bound $U^{(g)}$ of group g is computed using Equation (2.7) (line 25). Note that Equation (2.6) is used for the initial computation of the upper bound. If bound $U^{(g)}$ is lower than threshold $\sqrt{p_g}(1-\alpha)\lambda$, the parameters of the group are set to zeros by following Lemma 2.2 (lines 26 and 27). If the bound does not meet the condition, the same procedure as that of the original BCD is performed (lines 28–34). Next, $\|\beta^{(g)} - \tilde{\beta}^{(g)}\|_2$, which is used for the computation of the upper bound is updated (line 35).

In terms of the computation cost, our algorithm has the following property:

Theorem 2.1 (Computation cost) *Let S and S' be the ratios of the un-skipped groups to all the updates for all the parameter groups in Algorithm 2.1 when Lemma 2.2 and Equation (2.4) are used, respectively. Suppose that all groups have the same size, p_g. If t_m and t_f are the numbers of iterations of BCD for the main loop and element-level loop, respectively, our approach requires $\mathcal{O}(G\{(Q+St_m)(pp_g+p_g^2)+S'p_gt_m(t_fp_g+1)+Q\})$ or $\mathcal{O}(G\{(Q+St_m)np_g+S'p_gt_m(t_fp_g+1)+Q\})$ time.*

According to Theorem 2.1, when we have a large number of groups that are skipped on the basis of the upper bound, the rate of un-skipped groups S in Theorem 2.1 is small. As a result, the total computation cost is effectively reduced.

In terms of the objective value after convergence, our algorithm has the following property:

Theorem 2.2 *Suppose that the regularization constants $\lambda_q > 0$ in Algorithm 2.1 are the same as those of the original BCD. In addition, we assume that the BCD converges to a global optimal solution. Then, the solution of Algorithm 2.1 has the same value of the objective function as that of the original BCD.*

Theorem 2.2 shows that our algorithm returns the same value of the objective function as the original approach if BCD is not trapped on solutions other than global optimal solutions or does not cycle parameters. Furthermore, if the order of the updates in BCD in our algorithm is the same as the original BCD under the assumption in Theorem 2.2 , our method returns the same parameters and the objective value as those of the original BCD.

Algorithm 2.1 Fast Sparse Group Lasso

$\mathbb{A} = \{1, ..., G\}$, $\beta \leftarrow 0$, $\tilde{\beta} \leftarrow 0$; ▷ $\mathbb{A}$ has all the group indices
for each $g \in \mathbb{A}$ **do** ▷ The precomputation for the bounds
  for each $l \in \mathbb{A}$ **do**
    compute $\|\hat{K}^{(g)}[l]\|_2$;
for $q = 0$ to $Q - 1$ **do** ▷ The loop for the sequential rule
  $\mathbb{C} = \emptyset$; ▷ Initialize candidate group set $\mathbb{C}$
  for each $g \in \mathbb{A}$ **do** ▷ The loop for extracting candidate group set
    compute $C^{(g)}$ by Equation (2.8);
    if $C^{(g)} > \sqrt{p_g}(1-\alpha)\lambda_q$ **then** ▷ Add groups to $\mathbb{C}$ by following Lemma 2.5
      add g to $\mathbb{C}$;
  repeat ▷ The main loop for BCD on candidate group set $\mathbb{C}$
    for each $g \in \mathbb{C}$ **do** ▷ Group-level outer loop
      if $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda_l)\|_2 \leq \sqrt{p_g}(1-\alpha)\lambda_q$ **then** ▷ Check Equation (2.4)
        $\beta^{(g)} \leftarrow 0$;
      else
        repeat ▷ Element-level loop
          update $\beta^{(g)}$ by Equation (2.2);
        until $\beta^{(g)}$ converges
  until β converges
  repeat ▷ The main loop for BCD with the upper bounds
    $\tilde{\beta} \leftarrow \beta$; ▷ Set the reference parameter vector
    for each $g \in \mathbb{A}$ **do** ▷ The precomputation for the upper bounds
      compute $\|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2$;
    for each $g \in \mathbb{A}$ **do** ▷ Group-level outer loop
      compute $U^{(g)}$ by Equation (2.7);
      if $U^{(g)} \leq \sqrt{p_g}(1-\alpha)\lambda_q$ **then** ▷ Skip the group by following Lemma 2.2
        $\beta^{(g)} \leftarrow 0$;
      else
        if $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda_l)\|_2 \leq \sqrt{p_g}(1-\alpha)\lambda_q$ **then** ▷ Check Equation (2.4)
          $\beta^{(g)} \leftarrow 0$;
        else
          repeat ▷ Element-level loop
            update $\beta^{(g)}$ by Equation (2.2);
          until $\beta^{(g)}$ converges
      update $\|\beta^{(g)} - \tilde{\beta}^{(g)}\|_2$; ▷ For online update for the upper bounds
  until β converges

2.3.6 Extention: Overlapping Group Lasso and Graph Lasso

Our algorithm assumes that groups do not overlap each other. However, our algorithm is relatively easy to be extended for overlapping groups (Overlapping Group Lasso) and graphs (Graph Lasso) by using Latent Group Lasso (Jacob et al., 2009; Obozinski et al., 2011). It uses independent latent parameter vectors for each group to handle overlapping groups. The original parameter vector is represented as the summation of the latent parameter vectors which is called latent decomposition (Obozinski et al., 2011). To obtain the solution of the latent parameter vectors, we utlize covariate duplication that duplicates the parameters belonging to several groups (Jacob et al., 2009; Obozinski et al., 2011). Latent decomposition and covariate duplication allow us to use the same objective function as problem (2.1) for SGL with a different size of the problem. Therefore, since we can use the same updating rule of BCD for Latent Group Lasso as that of the original BCD for problem (2.1), our algorithm can be also used for Overlapping Group Lasso and Graph Lasso via Latent Group Lasso. By following (Obozinski et al., 2011), the problem setup of Latent Group Lasso is described below.

Let $X \in \mathbb{R}^{n \times p}$ be a matrix of features and groups be overlapped each other. We can represent X as $\tilde{X} \in \mathbb{R}^{n \times \sum_{g=1}^{G} p_g}$ by having a block matrix $X^{(g)} \in \mathbb{R}^{n \times p_g}$ for each group independently. Note that $\sum_{g=1}^{G} p_g = p$ holds when groups do not overlap. By using new matrix of features $\tilde{X}$, the linear regression model is represented as $y = \sum_{g=1}^{G} \tilde{X}^{(g)} v^{(g)}$ where $v^{(g)}$ is a latent parameter vector of group g. Latent Group Lasso (Jacob et al., 2009) uses $\lambda \sum_{g=1}^{G} \sqrt{p_g} \|v^{(g)}\|_2$ as the penalty for this model. Therefore, in the case of SGL with overlapping groups, solution $\hat{v} \in \mathbb{R}^{\sum_{g=1}^{G} p_g}$ is obtained by solving the following problem:

$$\hat{v} = \underset{v \in \mathbb{R}^{\sum_{g=1}^{G} p_g}}{\arg\min} \left(\frac{1}{2} \|y - \sum_{g=1}^{G} \tilde{X}^{(g)} v^{(g)}\|_2^2 + (1-\alpha)\lambda \sum_{g=1}^{G} \sqrt{p_g} \|v^{(g)}\|_2 + \alpha\lambda \|v\|_1 \right), \quad (2.10)$$

where $v \in \mathbb{R}^{\sum_{g=1}^{G} p_g}$ represents $v = [v^{(1)\mathrm{T}}, v^{(2)\mathrm{T}}, ..., v^{(G)\mathrm{T}}]^{\mathrm{T}}$. Problem (2.10) can be seen as an equation that has a different size from problem (2.1). Therefore, our algorithm can be used for problem (2.10) of overlapping groups. According to (Jacob et al., 2009), we can solve Graph Lasso that has graph structures in the feature vector by utilizing problem (2.10). For instance, if we have undirected graph (I, E) where I and E are features and edges among features respectively, we can consider the problem

that finds important edges from the graph. In this case, we replace the regularization term in problem (2.10) with $\lambda \sum_{e \in E} \sqrt{2} \|v^{(e)}\|_2$ where $v^{(e)} \in \mathbb{R}^2$ corresponds to two nodes (features). In other words, we treat an edge as a group and select important edges (groups) via problem (2.10). Although we cite the selection of edges as an example, we can also regard cliques and subgraphs as groups.

2.4 Related Work

To improve the efficiency of optimization with sparsity-inducing regularization, safe screening is generally used (Ghaoui et al., 2012); it eliminates zero parameters in the solution before the optimization is initiated. As the size of the feature vector can be reduced before entering the optimization, the efficiency of the optimization is improved. The current state-of-the-art safe screening method for SGL is the GAP Safe rule (Wang and Ye, 2014; Ndiaye et al., 2015; 2016; 2017), which is based on duality gap computation. The duality gap is computed as the difference between the primal and dual problems of SGL. They define a safe region that contains the solution based on the duality gap. By utilizing the safe region, this approach can identify groups and features that must be inactive, and eliminates them. If the safe region is small, this approach effectively eliminates groups and features. However, unless λ is large or a good approximate solution is already known, the screening is often ineffective (Johnson and Guestrin, 2016). To overcome this problem, Ndiaye et al. (Ndiaye et al., 2017) used the dynamic safe rule (Bonnefoy et al., 2014; 2015) with the GAP Safe rule for SGL. This dynamic GAP Safe rule effectively eliminates groups and features by repeatedly using the GAP Safe rule during the iterations of BCD.

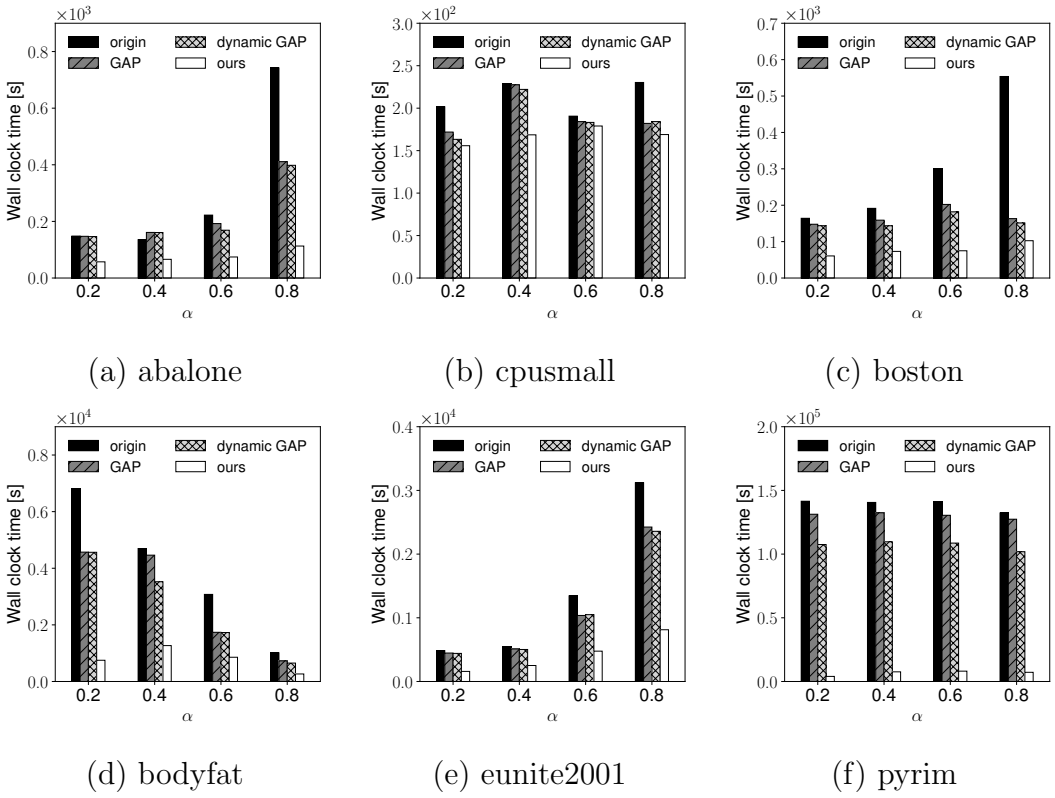
2.5 Experiments

We evaluated the processing time and prediction error of our approach by conducting experiments on six datasets from the LIBSVM[1] website (*abalone*, *cpusmall*, *boston*, *bodyfat*, *eunite2001*, and *pyrim*). The numbers of data points were 4177, 8192, 506, 252, 336, and 74, respectively. The number of features were 8, 12, 13, 14, 16, and

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`

27, respectively. In order to obtain group structure, we used the polynomial features of these datasets (Roth and Fischer, 2008). In particular, we created second-order polynomial features. The groups, which consisted of product over combinations of features up to the second degree, were created by using a polynomial kernel. For instance, if we had a pair of features (a, b), we created six features of $(1, a, b, ab, a^2, b^2)$ by using polynomial feature.[2] We handled these six features as one group and concatenated the original features and the feature groups by following the existing experimental setting of Group Lasso (Roth and Fischer, 2008). Since these groups are overlapped and the feature vector has duplicated features, we can say that this experimental setting uses latent decomposition and covariate duplication as we described in Section 2.3.6. In such a setting, we can use the original BCD and our method as shown in Section 2.3.6. As a result, the numbers of groups for each dataset were 36, 78, 91, 105, 136, and 378, respectively. The total numbers of features were 176, 408, 481, 560, 736, and 2133, respectively.

We compared our method with the original BCD, GAP Safe rule (Ndiaye et al., 2016), and dynamic GAP Safe rule (Ndiaye et al., 2017). We tuned λ for all approaches based on the sequential rule by following the methods in (Wang and Ye, 2014; Ndiaye et al., 2015; 2016; 2017). The search targets for hyperparameter λ was a non-increasing sequence of Q parameters $(\lambda_q)_{q=0}^{Q-1}$ defined as $\lambda_q = \lambda_{max} 10^{-\delta q/Q-1}$. We used $\delta = 4$ and $Q = 100$ (Wang and Ye, 2014; Ndiaye et al., 2015; 2016; 2017). For a fair comparison, λ_{max} was computed according to the dual norm by following the concept of GAP Safe rule (Ndiaye et al., 2016); GAP Safe rule safely eliminates groups and features under this setting. For dynamic GAP Safe rule, the interval of duality gap computations is set to 10 (Ndiaye et al., 2017). For another tuning parameter α, we used the settings $\alpha \in [0.2, 0.4, 0.6, 0.8]$. We stopped the algorithm for each λ_q when the relative tolerance $\|\beta - \beta_{new}\|_2 / \|\beta_{new}\|_2$ dropped below 10^{-5} for all approaches (Johnson and Guestrin, 2016; 2017). All the experiments were conducted on a Linux 2.20 GHz Intel Xeon server with 264 GB of main memory.



(a) abalone (b) cpusmall (c) boston

(d) bodyfat (e) eunite2001 (f) pyrim

Figure 2.1: Processing times of sequential rules for each hyperparameter α.

2.5.1 Processing Time

We evaluated the processing times of the sequential rules for each $\alpha \in [0.2, 0.4, 0.6, 0.8]$. Figure 2.1 shows the processing time of each approach on the six datasets. Note that the processing times include precomputation times for a fair comparison. In the figure, the terms *origin*, *GAP*, *dynamic GAP*, and *ours* represent the standard BCD, GAP Safe rule (Ndiaye et al., 2016), dynamic GAP Safe rule (Ndiaye et al., 2017), and our approach, respectively. Our approach is faster than the previous approaches for all datasets and hyperparameters; it reduces the processing time by up to 97% from the standard approach as shown in Figure 2.1 (f). Table 2.1 shows the number of computations of Equation (2.4), which is the main bottleneck of BCD. We can see that our method could reduce the number of the bottleneck computations. Furthermore, Figure 2.2 is the ratio of the number of the performed computations

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html`

Table 2.1: Numbers of computations of Eq. (2.4)

dataset	# of computations of Eq. (2.4)	
	origin	ours
abalone	1.141×10^5	$\mathbf{3.168 \times 10^3}$
cpusmall	2.105×10^5	$\mathbf{7.768 \times 10^4}$
boston	1.248×10^6	$\mathbf{9.998 \times 10^4}$
bodyfat	1.694×10^7	$\mathbf{2.403 \times 10^6}$
eunite2001	8.629×10^5	$\mathbf{2.052 \times 10^5}$
pyrim	7.667×10^7	$\mathbf{7.523 \times 10^6}$

of Equation (2.4) in our algorithm to that of the original BCD for each λ in the sequential rule with $\alpha = 0.2$ on boston dataset. K in the figure is the number of iterations of BCD. In particular, our method could effectively skip the computations for large λ. This is because $U^{(g)} \leq \sqrt{p_g}(1-\alpha)\lambda$ in Lemma 2.2 is more likely to hold for large λ. In addition, the number of the skipped computations is also large for large K because term $\|\beta^{(g)} - \tilde{\beta}^{(g)}\|_2$ in our upper bound becomes smaller as the parameters converge. As a result, the upper bound becomes tighter as the parameters converge and $U^{(g)} \leq \sqrt{p_g}(1-\alpha)\lambda$ in Lemma 2.2 is more likely to hold. These results suggest the effectiveness of the upper bound and candidate group set, which effectively reduce the number of computations, and contribute to the reduction of the processing time, as shown in Figure 2.1. The GAP Safe rule and dynamic GAP Safe rule eliminate groups and features that must be inactive, and increase the efficiency of BCD. However, when they cannot eliminate a significant number of groups and features, they require a large computation cost for BCD. To be specific, large numbers of groups and features remain when λ has a small value even if we use dynamic GAP Safe rule. This is because the safe region is large for small λ (Ndiaye et al., 2015; 2016), and it contains many groups and features that may be active. Furthermore, if the screening cannot eliminate a significant number of groups and features, the processing time may increase owing to the computation of the duality gap, as shown for $\alpha = 0.4$ in Figure 2.1 (a).

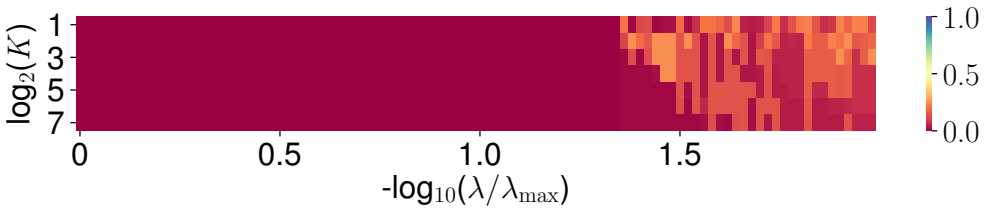


Figure 2.2: Ratio of performed computations of Eq. (2.4) for each λ in the sequential rule (boston dataset with $\alpha = 0.2$). K is the number of iterations of BCD.

2.5.2 Accuracy

In this section, we evaluate the value of the objective function on training data and the prediction error on test data to confirm the effectiveness of our algorithm. We split the data into training and test data for each dataset. That is, 50% of a dataset was used as training data for evaluating the objective value and the other 50% of a dataset was used as test data for evaluating the prediction error in terms of the squared loss for the response. The results of the objective value and the prediction error are shown in Tables 2.2 and 2.3, respectively. The objective values and the prediction errors of our approach are the same as those of the original approach. The results presented in Tables 2.2 and 2.3 indicate that our method can achieve the same accuracy as that of the original approach while improving the efficiency. It should be noted that the solutions obtained by the original BCD and the proposed method can be different. This is because the objective function is not necessarily strongly convex function and the updating orders are different between the original BCD and the proposed method. In fact, Theorem 2.2 only guarantees the consistency of the objective values between the original BCD and the proposed method under the assumption of convergence to a global optimal solution. If the updating order in our algorithm is the same as the original BCD, our method returns the same parameters and the objective value as those of the original BCD.

2.5.3 Overlapping Group Lasso and Graph Lasso

We show the processing times of our algorithm in the case of Overlapping Group Lasso and Graph Lasso on artificial datasets. We generated artificial datasets by following (Ndiaye et al., 2016) and (Jacob et al., 2009): y was generated by following

Table 2.2: Values of objective function after convergence.

dataset	objective value	
	origin	ours
abalone	**2.209**	**2.209**
cpusmall	**7.862**	**7.862**
boston	**9.613**	**9.613**
bodyfat	$\mathbf{5.284 \times 10^{-3}}$	$\mathbf{5.284 \times 10^{-3}}$
eunite2001	$\mathbf{2.004 \times 10^{2}}$	$\mathbf{2.004 \times 10^{2}}$
pyrim	$\mathbf{4.568 \times 10^{-3}}$	$\mathbf{4.568 \times 10^{-3}}$

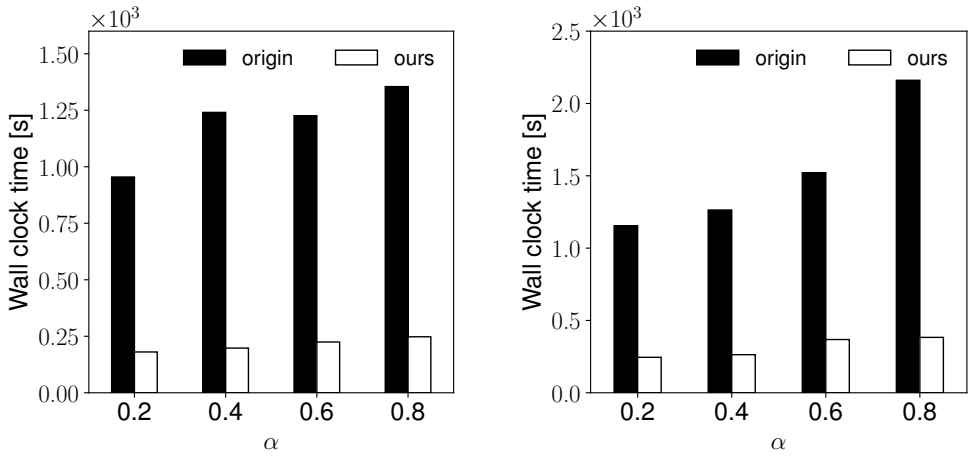
Table 2.3: Prediction errors.

dataset	prediction error	
	origin	ours
abalone	**2.232**	**2.232**
cpusmall	**7.886**	**7.886**
boston	**9.887**	**9.887**
bodyfat	$\mathbf{5.434 \times 10^{-3}}$	$\mathbf{5.434 \times 10^{-3}}$
eunite2001	$\mathbf{2.010 \times 10^{2}}$	$\mathbf{2.010 \times 10^{2}}$
pyrim	$\mathbf{4.615 \times 10^{-3}}$	$\mathbf{4.615 \times 10^{-3}}$

$y = X\beta + 0.01\epsilon$ where $\epsilon \sim N(0, nI)$, and X followed a multivariate normal distribution such that the correlation between i-th and j-th features was $0.5^{|i-j|}$. Each element of β was generated from $\text{sign}(\xi) \times U$ where U and ξ follow uniform distributions such that $U \sim [0.5, 10]$ and $\xi \sim [-1, 1]$, respectively.

For Overlapping Group Lasso, each size of groups was 10, and we set $p = 1000, n = 100$. We generated two datasets whose sizes of the overlaps were 2 and 5, respectively: when the size of overlap is 2, indices of the groups are represented as $\{1, ..., 10\}, \{9, ..., 18\}, ...$

For Graph Lasso, we generated datasets such that each edge corresponds to each group (Jacob et al., 2009). We set $p = 100, n = 100$, and used chain graph by folowing (Jacob et al., 2009). We generated two datasets whose lengths of the chains were 2 and 4, respectively. The other settings such as hyperparameters were the

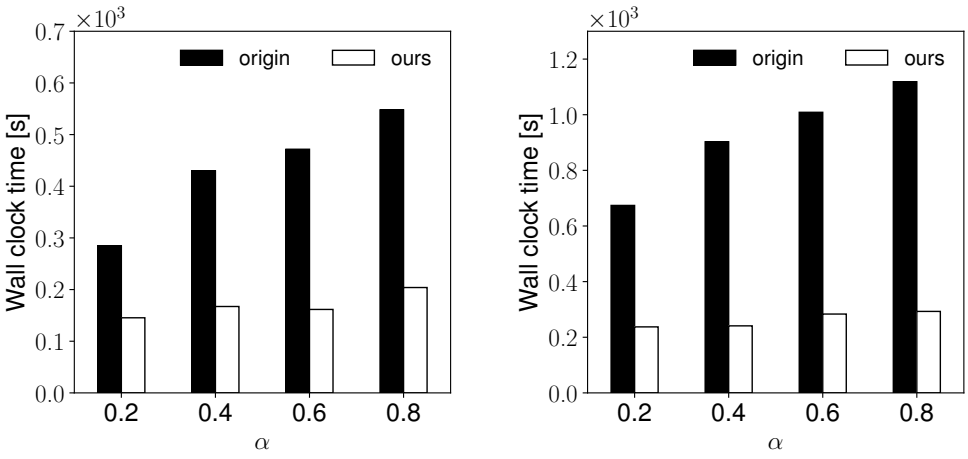


(a) overlapping=2 (b) overlapping=5

Figure 2.3: Processing times of sequential rules of Overlapping Group Lasso for each hyperparameter α.

same as those in the previous experiments.

Figures 2.3 and 2.4 are the processing times of Overlapping Group Lasso and Graph Lasso, respectively. Our algorithm is faster than the original algorithm in both cases. The results suggest that our algorithm can be extended to Overlapping Group Lasso and Graph Lasso on the basis of overlap norm (Jacob et al., 2009), and speed up these methods.



(a) chain=2 (b) chain=4

Figure 2.4: Processing times of sequential rules of Graph Lasso for each hyperparameter α.

2.6 Discussion

In this chapter, we proposed a fast BCD for SGL. Although our method skips unnecessary updates by utilizing the upper bound, the bound will be loose if we simply take an upper bound of $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ by using the Cauchy–Schwarz inequality and the triangle inequality. The point of our bound is to incorporate difference of the parameter vectors $\|\beta^{(g)} - \tilde{\beta}^{(g)}\|_2$ in Equation (2.6). The term of $\|\beta^{(g)} - \tilde{\beta}^{(g)}\|_2$ becomes increasingly smaller during the optimization because we regularly update $\tilde{\beta}^{(g)}$ as described in line 21 of Algorithm 2.1. Since we will finally obtain $\|\beta^{(g)} - \tilde{\beta}^{(g)}\|_2 = 0$ and $\Lambda(\cdot, \cdot) = 0$ in Equation (2.6) if the parameter vector converges to a global optimum solution, we expect the bound to be tight during the optimization. It is also worth mentioning that the upper bound of Equation (2.6) is not the tightest upper bound. In fact, we can replace $\Lambda(g, g)$ with $-\Lambda(g, g)$ in Equation (2.6) to derive a tighter bound than Equation (2.6). In this way, we expect our method to be further accelerated by finding a tighter upper bound than Equation (2.6).

2.7 Summary

We proposed a fast Block Coordinate Descent for Sparse Group Lasso. The main bottleneck of the original Block Coordinate Descent is the computation to check whether groups have zero or nonzero parameter vectors, because it uses all the parameters or data points. In contrast, our approach identifies the groups whose parameters *must be updated to zeros* by using the parameters in the group, and skips the computation. Furthermore, the proposed approach identifies the candidate group set, which contains the groups whose parameters *must not be updated to zeros*. The parameters are preferentially updated in the set to raise the effectiveness of Block Coordinate Descent. The attractive point of our method is that it does not need any additional hyperparameters. In addition, it provably guarantees the same results as the original method. The experimental results showed that our method reduces the processing time by up to 97% without any loss of accuracy compared with that of the original method. We also showed that our method can be extended to Overlapping Group Lasso and Graph Lasso; it could be used for various wide linear regression models with structured sparsity-inducing norms.

2.8 Proofs

Proof of Lemma 2.1

Before we prove the above lemma, we prove the following lemmas:

Lemma 2.7 *For each $\|S(X^{(g)\mathrm{T}}r_{(-g)}, \alpha\lambda)\|_2$ of group g, we have $\|X^{(g)\mathrm{T}}r_{(-g)}\|_2 \geq \|S(X^{(g)\mathrm{T}}r_{(-g)}, \alpha\lambda)\|_2$.*

Proof *We have*

$$\|S(z, \alpha\lambda)\|_2 = \|\mathrm{sign}(z)(|z| - \alpha\lambda)_+\|_2 = \|(|z| - \alpha\lambda)_+\|_2. \tag{2.11}$$

In addition, we have

$$\|(|z| - \alpha\lambda)_+\|_2 \leq \||z|\|_2 = \|z\|_2, \tag{2.12}$$

because $0 \leq (|z| - \alpha\lambda)_+ \leq |z|$. From Equations (2.11) and (2.12), we have $\|S(z, \alpha\lambda)\|_2 \leq \|z\|_2$. Because $z = X^{(g)\mathrm{T}}r_{(-g)}$, we achieve the inequality of Lemma 2.7. □

Now, we prove Lemma 2.1:

Proof *Let $K^{(g,:)} \in \mathcal{R}^{p_g \times p}$ be a block matrix of K that corresponds to group g. We introduce notations $R^{(g)} := X^{(g)\mathrm{T}}r_{(-g)}$ and $\tilde{R}^{(g)} := X^{(g)\mathrm{T}}\tilde{r}_{(-g)}$ for simplicity. From Equation (2.5), we have*

$$R^{(g)} = X^{(g)\mathrm{T}}y - K^{(g,:)}\beta + K^{(g,g)}\beta^{(g)}.$$

This equation is transformed to the following form:

$$\begin{aligned} &X^{(g)\mathrm{T}}y - K^{(g,:)}\tilde{\beta} + K^{(g,g)}\tilde{\beta}^{(g)} + K^{(g,g)}\Delta\beta^{(g)} - K^{(g,:)}\Delta\beta \\ &= \tilde{R}^{(g)} + K^{(g,g)}\Delta\beta^{(g)} - \textstyle\sum_{l=1}^{G} K^{(g,l)}\Delta\beta^{(l)}, \end{aligned}$$

where $\Delta\beta^{(g)} = \beta^{(g)} - \tilde{\beta}^{(g)}$ and $\Delta\beta = \beta - \tilde{\beta}$. From the aforementioned equation and the triangle inequality, we obtain the following inequality:

$$\|R^{(g)}\|_2 \leq \|\tilde{R}^{(g)}\|_2 + \|K^{(g,g)}\Delta\beta^{(g)}\|_2 + \textstyle\sum_{l=1}^{G}\|K^{(g,l)}\Delta\beta^{(l)}\|_2. \tag{2.13}$$

In the second and third terms on the right-hand side of Equation (2.13), the i-th element of $K^{(g,l)}\Delta\beta^{(l)}$ is computed as the inner product of $K^{(g,l)}[i,:]\Delta\beta^{(l)}$. We then obtain the following upper bound for the absolute value of the inner product by using the Cauchy–Schwarz inequality:

$$|K^{(g,l)}[i,:]\Delta\beta^{(l)}| \leq \|K^{(g,l)}[i,:]\|_2 \|\Delta\beta^{(l)}\|_2. \tag{2.14}$$

In addition, because $\|\Delta\beta^{(l)}\|_2$ is a scalar, we obtain the following inequality for $\|K^{(g,l)}\Delta\beta^{(l)}\|_2$ in Equation (2.13) by using Equation (2.14):

$$\|K^{(g,l)}\Delta\beta^{(l)}\|_2 \leq \|\hat{K}^{(g)}[l]\|_2 \|\Delta\beta^{(l)}\|_2 = \Lambda(g,l). \tag{2.15}$$

From Equations (2.13) and (2.15), we obtain the following upper bound:

$$\|R^{(g)}\|_2 \leq \|\tilde{R}^{(g)}\|_2 + \Lambda(g,g) + \textstyle\sum_{l=1}^{G} \Lambda(g,l) = U^{(g)}.$$

Finally, we obtain the following upper bound by using the aforementioned inequality and Lemma 2.7:

$$\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 \leq \|X^{(g)\mathrm{T}} r_{(-g)}\|_2 = \|R^{(g)}\|_2 \leq U^{(g)},$$

which completes the proof. □

Proof of Lemma 2.2

Proof *We have $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 \leq U^{(g)} \leq \sqrt{p_g}(1-\alpha)\lambda$ from Lemma 2.1 if $U^{(g)} \leq \sqrt{p_g}(1-\alpha)\lambda$. Because $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 \leq \sqrt{p_g}(1-\alpha)\lambda$ holds for group g, $\beta^{(g)}$ is the zero vector from Equation (2.4).* □

Proof of Lemma 2.3

Proof *For the computation of Equation (2.6), we can precompute $\|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2$ and $\|\hat{K}^{(g)}[\cdot]\|_2$ before entering the group-level loop and main loop, respectively. Thus, we can obtain the terms $\|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2$, $\|\hat{K}^{(g)}[\cdot]\|_2$, and $\Lambda(g,l)$ in Equation (2.6) in $\mathcal{O}(1)$ time. If $\beta^{(g)}$ is updated to $\beta^{(g)'}$, the upper bound of Equation (6) can be updated by using Equation (2.7). It needs $\mathcal{O}(p_g)$ time because $\|\beta^{(g)'} - \tilde{\beta}^{(g)}\|_2$ is computed in*

$\mathcal{O}(p_g)$ time, and we can obtain $\Lambda(g,g)$ and $\|\hat{K}^{(g)}[g]\|_2\|$ in $\mathcal{O}(1)$ time, just as described. Therefore, we can compute the upper bound of Equation (2.6) in $\mathcal{O}(p_g)$ time for group g by using Equation (2.7). □

Proof of Lemma 2.4

Before we prove the above lemma, we introduce the following definitions and lemmas:

Definition 2.5 (Lower bound) *Let $L^{(g)}$ be a lower bound of $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ in Equation (2.4) and $\tilde{r}_{(-g)}$ be a partial residual of Equation (2.3) before entering the group-level outer loop. Then, $L^{(g)}$ is defined as follows:*

$$L^{(g)} = \|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2 - \Lambda(g,g) - \textstyle\sum_{l=1}^{G} \Lambda(g,l) - \alpha\lambda\sqrt{2p_g}, \tag{2.16}$$

where $\Lambda(g,l) = \|\hat{K}^{(g)}[l]\|_2 \|\beta^{(l)} - \tilde{\beta}^{(l)}\|_2$. In the above equation, the i-th element of $\hat{K}^{(g)}[l] \in \mathcal{R}^{p_g}$ is given as $\|K^{(g,l)}[i,:]\|_2$, that is, the l_2 norm of the i-th row vector in block matrix $K^{(g,l)} \in \mathcal{R}^{p_g \times p_l}$ of $K := X^{\mathrm{T}} X \in \mathcal{R}^{p \times p}$. $\tilde{\beta}^{(g)}$ is a parameter vector before entering the group-level outer loop.

Lemma 2.8 *For each $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ of group g, we have $L^{(g)}$ $\leq \|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$.*

Proof *Let $I^{+}_{(g)} \in \{0,1\}^{p_g}$ be a vector whose i-th element takes 1 if the absolute value of i-th element in $X^{(g)\mathrm{T}} r_{(-g)}$ is greater than $\alpha\lambda$, and takes 0 if the absolute value of the element is less than or equal to $\alpha\lambda$. We then have*

$$\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 = \||X^{(g)\mathrm{T}} r_{(-g)}| - \alpha\lambda I^{+}_{(g)} - |X^{(g)\mathrm{T}} r_{(-g)}| \odot (1 - I^{+}_{(g)})\|_2. \tag{2.17}$$

According to Equation (2.17) and the triangle inequality, we have

$$\begin{aligned} &\||X^{(g)\mathrm{T}} r_{(-g)}| - \alpha\lambda I^{+}_{(g)} - |X^{(g)\mathrm{T}} r_{(-g)}| \odot (1 - I^{+}_{(g)})\|_2 \\ &\geq \|X^{(g)\mathrm{T}} r_{(-g)}\|_2 - \|\alpha\lambda I^{+}_{(g)}\|_2 - \|X^{(g)\mathrm{T}} r_{(-g)} \odot (1 - I^{+}_{(g)})\|_2. \end{aligned} \tag{2.18}$$

Let p^{+}_{g} be the number of elements, taking the value 1 in $I^{+}_{(g)}$. Then, we clearly have

$$\|\alpha\lambda I^{+}_{(g)}\|_2 = \alpha\lambda\sqrt{p^{+}_{g}}. \tag{2.19}$$

In addition, because the absolute values of the elements in $X^{(g)\mathrm{T}} r_{(-g)} \odot (1 - I^{+}_{(g)})$ *is less than or equal to* $\alpha\lambda$*, we have*

$$\|X^{(g)\mathrm{T}} r_{(-g)} \odot (1 - I^{+}_{(g)})\|_2 \leq \|\alpha\lambda(1 - I^{+}_{(g)})\|_2 = \alpha\lambda\sqrt{p_g - p_g^{+}}. \tag{2.20}$$

From Equations (2.17), (2.18), (2.19), and (2.20), we have

$$\begin{aligned}\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 &\geq \|X^{(g)\mathrm{T}} r_{(-g)}\|_2 - \alpha\lambda\sqrt{p_g^{+}} - \alpha\lambda\sqrt{p_g - p_g^{+}} \\ &= \|X^{(g)\mathrm{T}} r_{(-g)}\|_2 - \alpha\lambda(\sqrt{p_g^{+}} + \sqrt{p_g - p_g^{+}}) \\ &\geq \|X^{(g)\mathrm{T}} r_{(-g)}\|_2 - \alpha\lambda\sqrt{2p_g}. \end{aligned} \tag{2.21}$$

We use the inequality of $\sqrt{p_g^{+}} + \sqrt{p_g - p_g^{+}} \leq 2\sqrt{p_g/2}$ *in Equation (2.21). Next, we derive the lower bound of* $\|X^{(g)\mathrm{T}} r_{(-g)}\|_2$ *in Equation (2.21). Similar to Equation (2.13) in the proof of the upper bound, we have the following inequality by using the triangle inequality:*

$$\|X^{(g)\mathrm{T}} r_{(-g)}\|_2 = \|R^{(g)}\|_2 \geq \|\tilde{R}^{(g)}\|_2 - \|K^{(g,g)}\Delta\beta^{(g)}\|_2 - \textstyle\sum_{l=1}^{G} \|K^{(g,l)}\Delta\beta^{(l)}\|_2. \tag{2.22}$$

From Equations (2.22) and (2.15), we obtain the following lower bound:

$$\|X^{(g)\mathrm{T}} r_{(-g)}\|_2 \geq \|\tilde{R}^{(g)}\|_2 - \Lambda(g,g) - \textstyle\sum_{l=1}^{G} \Lambda(g,l). \tag{2.23}$$

Finally, we obtain the following lower bound by using Equations (2.21) and (2.23):

$$\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 \geq \|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2 - \Lambda(g,g) - \textstyle\sum_{l=1}^{G} \Lambda(g,l) - \alpha\lambda\sqrt{2p_g} = L^{(g)}, \tag{2.24}$$

which completes the proof. □

The following lemma shows that we can identify groups whose parameters must be updated to nonzeros by using the lower bound:

Lemma 2.9 (Groups with nonzero vectors) *If we have* $L^{(g)} > \sqrt{p_g}(1 - \alpha)\lambda$*, parameter* $\beta^{(g)}$ *for group* g *is a nonzero vector.*

Proof *We have* $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 \geq L^{(g)} > \sqrt{p_g}(1 - \alpha)\lambda$ *from Lemma 2.8 if* $L^{(g)} > \sqrt{p_g}(1-\alpha)\lambda$*. Because* $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 > \sqrt{p_g}(1 - \alpha)\lambda$ *holds for group* g*,* $\beta^{(g)}$ *is the nonzero vector from Equation (2.4).* □

If we have $L^{(g)} \leq \sqrt{p_g}(1-\alpha)\lambda$, it is clear that parameter $\beta^{(g)}$ for group g can have a zero vector. Now, we prove Lemma 2.4:

Proof *From Lemma 2.1 and 2.8, we have*

$$L^{(g)} \leq \|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2 \leq U^{(g)}. \tag{2.25}$$

In addition, we also have

$$L^{(g)} \leq C^{(g)} \leq U^{(g)}, \tag{2.26}$$

because

$$\tfrac{L^{(g)}+U^{(g)}}{2} = \|X^{(g)\mathrm{T}} \tilde{r}_{(-g)}\|_2 - \alpha\lambda\sqrt{p_g/2} = C^{(g)}. \tag{2.27}$$

Thus, we have $L^{(g)} = C^{(g)} - \Lambda(g,g) - \sum_{l=1}^{G} \Lambda(g,l) - \alpha\lambda\sqrt{p_g/2}$, *and* $U^{(g)} = C^{(g)} + \Lambda(g,g) + \sum_{l=1}^{G} \Lambda(g,l) + \alpha\lambda\sqrt{p_g/2}$. *Therefore, the exact value of* $\|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2$ *exists within* $C^{(g)} \pm \epsilon$, *where* $\epsilon = \Lambda(g,g) + \sum_{l=1}^{G} \Lambda(g,l) + \alpha\lambda\sqrt{p_g/2}$. *In other words, we have*

$$|C^{(g)} - \|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2| \leq \epsilon. \tag{2.28}$$

For the upper bound, we clearly obtain the following inequality from Equation (2.25):

$$|U^{(g)} - \|S(X^{(g)\mathrm{T}} r_{(-g)}, \alpha\lambda)\|_2| \leq |U^{(g)} - L^{(g)}| = 2\epsilon, \tag{2.29}$$

which completes the proof. □

Proof of Lemma 2.5

Proof *From Equation (2.26), we have*

$$L^{(g)} \leq C^{(g)}. \tag{2.30}$$

By following Definition 2.4, we add a group to the candidate group set when the condition of $C^{(g)} > \sqrt{p_g}(1-\alpha)\lambda$ *holds. Then, we can consider two situations for the group from Equation (2.30): (i)* $C^{(g)} \geq L^{(g)} > \sqrt{p_g}(1-\alpha)\lambda$ *and (ii)* $C^{(g)} >$

$\sqrt{p_g}(1-\alpha)\lambda \geq L^{(g)}$. *In the case of (i), the group must have a nonzero parameter vector based on Lemma 2.9. In the case of (ii), the group can clearly have a nonzero parameter vector according to Lemma 2.9. Therefore, the candidate group set contains the groups whose parameters must be nonzeros because case (i) is included in the condition of* $C^{(g)} > \sqrt{p_g}(1-\alpha)\lambda$ *in Definition 2.4.* □

The candidate group set contains a subset of the nonzero groups *identified by using the lower bound of Lemma 2.9*. The lower bound confidently identifies groups with *nonzero* vectors while the upper bound of Lemma 1 identifies groups with *zero* vectors.

Proof of Lemma 2.6

Proof *Equation (2.8) can be clearly computed at O(1) time if we pre-compute* $\|X^{(g)\mathrm{T}}\tilde{r}_{(-g)}\|_2$. *Therefore, we require O(G) time to extract the candidate group set because we compute Equation (2.8) for all the groups.* □

Proof of Theorem 2.1

Proof *In Algorithm 1, first,* $\|\hat{K}^{(g)}[l]\|_2$ *is precomputed, which requires* $\mathcal{O}(p_g)$ *time. For all groups, the precomputation requires* $\mathcal{O}(p)$ *time. Before entering the group-level loop,* $\|X^{(g)\mathrm{T}}\tilde{r}_{(-g)}\|_2$ *is precomputed in* $\mathcal{O}(GQ(pp_g + p_g^2))$ *or* $\mathcal{O}(GQnp_g)$ *time to obtain the upper bounds. We can update the upper bounds when a parameter vector in a group is updated according to Definition 2.2. Thus, the updates of upper bounds require* $\mathcal{O}(GS'p_g)$ *time for a group-level loop by following Lemma 2.3. If the group is not skipped with respect to the upper bound, Equation (2.4) is computed in* $\mathcal{O}(pp_g+p_g^2)$ *time. Because the unskipped rate is S, the computation cost is* $\mathcal{O}(GS(pp_g + p_g^2))$ *or* $\mathcal{O}(GSnp_g)$ *time. If the group is not skipped with respect to Equation (2.4), the parameter vector is updated using Equation (2.2). Equation (2.2) requires* $\mathcal{O}(p_g^2)$ *time because* $\|S(Z^{(g)}, t\alpha\lambda)\|_2$ *in Equation (2.4) requires* $\mathcal{O}(p_g)$ *time, and* $Z^{(g)}$ *can be obtained in* $\mathcal{O}(p_g^2)$ *time based on precomputations. Because the unskipped rate is* S' *and the number of iterations for the element-level loop is* t_f, *the computation requires* $\mathcal{O}(GS't_f p_g^2)$ *time. Because the number of iterations for the main loop of BCD is* t_m, *the total computation cost of our approach is* $\mathcal{O}(G\{(Q+St_m)(pp_g+p_g^2)+S'p_g t_m(t_f p_g+1)+Q\})$ *or* $\mathcal{O}(G\{(Q+St_m)np_g + S'p_g t_m(t_f p_g+1)+Q\})$ *time.* □

Proof of Theorem 2.2

Proof *Since we assume that BCD converges to a global optimal solution, our algorithm converges at line 19 of Algorithm 2.1. This is because the lines 11–19 perform the original BCD on the parameter groups corresponding to the candidate group set. Although the lines 20–36 perform BCD with the upper bound, the parameter vector of group g must be a zero vector if $U^{(g)} \leq \sqrt{p_g}(1-\alpha)\lambda$ holds from Lemma 2.2. Thus, we safely skip the computation of such groups. In other cases, the parameter vector can be a nonzero vector, and Algorithm 2.1 does not skip the computation of such groups. Therefore, since BCD of lines 20–36 safely skips unnecessary updates and uses all the parameter vectors, it converges to a global optimal solution at line 36 of Algorithm 2.1 under our assumption of convergence of the original BCD and achieves the same objective value as that of the original BCD after convergence.* □

We note that Theorem 2.2 holds regardless of the group updating order because it guarantees *the converged value of the objective function* rather than *the converged parameter vector* under our assumption of convergence to a global optimal solution. Since the problem of SGL is either convex or strongly convex depending on the condition of X, the *optimal value of the objective function* is unique (while *the optimal parameter vector* may be non-unique).

We also note that if we assume that the order of updates for our method is the same as that of the original BCD, our algorithm returns the same parameter vectors and objective value as those of the original BCD. This property obviously holds since our method uses the same updating rule for parameter vectors as that of the original BCD and safely skips unnecessary updates.

Chapter 3

Fast CUR Matrix Decomposition for Regularized Self-Representation Model

3.1 Introduction

Matrix decomposition is a fundamental tool of machine learning, and it is used to decompose a data matrix into its low-rank approximation. Among various matrix decomposition methods such as Singular Value Decomposition (SVD), CUR matrix decomposition (CUR) (Mahoney and Drineas, 2009) has been a popular method due to its high interpretability. This is because the decomposed matrices consist of the subsets of the columns and the rows of the original data matrix. Namely, the decomposed matrices preserve the original elements in the data matrix. Thanks to its high interpretability, CUR has been successfully applied to a large number of domains, including gene expression data (Bien et al., 2010), network traffic data (Tong et al., 2008), bibliographic data (Sun et al., 2007), collaborative filtering (Mackey et al., 2011), hyperspectral medical image (Mahoney et al., 2006), and text data (Drineas et al., 2008).

In the literature, randomized algorithms (Mahoney and Drineas, 2009; Sun et al., 2007; Drineas et al., 2006; Tong et al., 2008) and deterministic algorithms (Bien et al., 2010; Mairal et al., 2011; Papailiopoulos et al., 2014) were proposed for CUR.

Although randomized algorithms were proposed first, they would be disconcerting to the practitioners as they obtain a different result on every run (Bien et al., 2010; Mairal et al., 2011). Specifically, when the sizes of the decomposed matrices are small, they can obtain a poor approximate result because the variance of the approximate results is large (Sun et al., 2007; Bien et al., 2010). To overcome these drawbacks, researchers have focused on deterministic algorithms that utilize a sparse optimization approach (Bien et al., 2010; Mairal et al., 2011). In deterministic algorithms, matrix decomposition is seen as a convex optimization problem with sparsity-inducing norms on the basis of group Lasso (Yuan and Lin, 2006). The model used in this approach is known as regularized self-representation (RSR) (Zhu et al., 2015). In particular, they optimize an objective with respect to the parameter vectors corresponding to the columns and the rows of the data matrix. Since the unimportant parameter vectors are turned into zero vectors by the sparsity-inducing norms, they can deterministically obtain the important columns and rows, which are used to construct the decomposed matrices.

Although deterministic algorithms are attractive, they suffer from high computation costs. In order to optimize the objective, they usually use coordinate descent, which iteratively updates each parameter vector corresponding to each column and row of the data matrix (Bien et al., 2010). Unfortunately, the computation cost of updating a parameter vector is quadratic with respect to the number of columns or rows of the data matrix. In addition, they need to iteratively update all the parameters until convergence. For the aforementioned reasons, deterministic algorithms require longer processing times as the sizes of data matrices increase.

This chapter proposes a fast deterministic algorithm for CUR. Our approach utilizes two ideas to speed up the deterministic CUR. The first idea is to safely skip the updates of the parameters in coordinate descent. Since a high computation cost is needed for one update in coordinate descent, we can effectively reduce the total computation cost by skipping the updates. Specifically, we identify the rows and columns whose parameters *must be updated to zeros* in linear time with respect to the number of columns or rows by approximately evaluating the necessary and sufficient conditions for the parameters to be zeros. The second idea is to preferentially update the parameters that *must be updated to nonzeros*. Because these nonzero parameters would correspond to the important columns and rows for the construction of the

decomposed matrices, our algorithm is expected to effectively optimize the objective function. Similarly to the computation in the first idea, it can identify the columns and rows whose parameters must be nonzeros in linear time. Another advantage of our algorithm is that it does not have additional hyperparameters, which incur additional computation costs for the tuning. Theoretically, our algorithm provably guarantees convergence to the same value of the objective function as that of the original algorithm. Experiments show that our method is up to 10× faster than the original method, and up to 4× faster than the state-of-the-art method while achieving the same accuracy.

3.2 Preliminary

In this section, we first explain the deterministic CUR by following (Bien et al., 2010; Mairal et al., 2011). Next, we describe coordinate descent, which optimizes the objective of deterministic CUR. Throughout the chapter, given a matrix $\mathbf{A}$, $\mathbf{A}_{(i)}$ and $\mathbf{A}^{(i)}$ denote the i-th row vector and i-th column vector of $\mathbf{A}$, respectively. Similarly, given a set of indices $\mathcal{I}$, $\mathbf{A}_{\mathcal{I}}$ and $\mathbf{A}^{\mathcal{I}}$ denote the submatrices of $\mathbf{A}$ containing only rows and columns in $\mathcal{I}$, respectively. $\|\mathbf{A}\|_F$ represents the Frobenius norm of matrix $\mathbf{A}$.

3.2.1 Deterministic CUR Matrix Decomposition

CUR provides a low-rank approximation to a data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$. In particular, CUR decomposes the data matrix $\mathbf{X}$ into the form of a product of three matrices as $\mathbf{X} \approx \mathbf{CUR}$, where $\mathbf{C} \in \mathbb{R}^{n \times c}$, $\mathbf{U} \in \mathbb{R}^{c \times r}$, and $\mathbf{R} \in \mathbb{R}^{r \times p}$. Unlike other low-rank approximations such as Singular Value Decomposition (SVD), CUR extracts $\mathbf{C}$ and $\mathbf{R}$ as small numbers of the column and row vectors of $\mathbf{X}$, respectively. In other words, columns of $\mathbf{C}$ and rows of $\mathbf{R}$ are subsets of the columns and the rows of the original data matrix $\mathbf{X}$, respectively. This property helps practitioners to interpret the result more easily than that in the case of SVD (Tong et al., 2008). If we need $\mathbf{U}$ after selecting $\mathbf{C}$ and $\mathbf{R}$, $\mathbf{U}$ is given by computing $\mathbf{C}^{+}\mathbf{X}\mathbf{R}^{+}$ where $\mathbf{A}^{+}$ represents the Moore-Penrose generalized inverse of matrix $\mathbf{A}$.

To select $\mathbf{C}$ or $\mathbf{R}$, Bien et al. (2010) utilized a sparse optimization approach. In particular, the selection of $\mathbf{C}$ or $\mathbf{R}$ from $\mathbf{X}$ can be seen as a convex optimization

Algorithm 3.1 Deterministic CUR

$\mathbb{A} = \{1, ..., p\}$, $\mathbf{W} \leftarrow \mathbf{0}$
repeat
  for each $i \in \mathbb{A}$ **do**
    update $\mathbf{W}_{(i)}$ by Equation (3.2);
until $\mathbf{W}$ converges

problem with sparsity-inducing norms. For the selection of $\mathbf{C}$, the optimization problem is defined as follows:

$$\min_{\mathbf{W} \in \mathbb{R}^{p \times p}} (\tfrac{1}{2} \|\mathbf{X} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda \textstyle\sum_{i=1}^{p} \|\mathbf{W}_{(i)}\|_2), \tag{3.1}$$

where $\mathbf{W} \in \mathbb{R}^{p \times p}$ is the parameter matrix, and $\lambda \geq 0$ is a regularization constant. This formulation is also known as regularized self-representation (RSR) (Zhu et al., 2015). The term $\|\mathbf{W}_{(i)}\|_2$ induces $\mathbf{W}_{(i)}$ to be a zero vector; this sparsity-inducing norm is also used in group Lasso (Yuan and Lin, 2006). The regularization constant λ controls the degree of sparsity of the parameter matrix $\mathbf{W}$. If $\mathbf{W}_{(i)}$ is a zero vector, the corresponding column of the data matrix $\mathbf{X}^{(i)}$ can be considered as an unimportant column for problem (3.1). On the other hand, $\mathbf{X}^{(i)}$ is important when the corresponding $\mathbf{W}_{(i)}$ is a nonzero vector. Therefore, we can select columns $\mathbf{C}$ as $\mathbf{X}^{\mathcal{I}}$, where $\mathcal{I} \subseteq \{1, ..., p\}$ represents the indices corresponding to the nonzero row vectors of $\mathbf{W}$. In this case, we obtain $\mathbf{U} = \mathbf{W}_{\mathcal{I}}$ or $\mathbf{U} = (\mathbf{X}^{\mathcal{I}})^{+}\mathbf{X}$. We note that although the above problem handles the selection of $\mathbf{C}$, Mairal et al. (2011) naturally extended the problem to the simultaneous selection of both $\mathbf{R}$ and $\mathbf{C}$. Throughout the chapter, we handle problem (3.1) focusing on simplicity; however, our approach can be easily applied to the problem of (Mairal et al., 2011) as described in Section 3.3.6.

3.2.2 Coordinate Descent

Problem (3.1) is simply solved by using coordinate descent (Bien et al., 2010). The algorithm iteratively updates each parameter vector $\mathbf{W}_{(i)}$ corresponding to each row of the parameter matrix $\mathbf{W}$ until $\mathbf{W}$ converges. Suppose that $\mathbf{X}^{(i)}$ is normalized as

$\|\mathbf{X}^{(i)}\|_2 = 1$. Then, the following equation is used to update $\mathbf{W}_{(i)}$:

$$\mathbf{W}_{(i)} = (1 - \lambda / \|\mathbf{z}_i\|_2)_+ \mathbf{z}_i. \tag{3.2}$$

In the equation, $\mathbf{z}_i \in \mathbb{R}^{1 \times p}$ is computed as follows:

$$\mathbf{z}_i = \mathbf{X}^{(i)\mathrm{T}} (\mathbf{X} - \textstyle\sum_{j \neq i}^{p} \mathbf{X}^{(j)} \mathbf{W}_{(j)}). \tag{3.3}$$

Algorithm 3.1 shows the pseudocode of coordinate descent. The inner loop (lines 3–4) performs Equation (3.2) to update each row of $\mathbf{W}$, and the outer loop (lines 2–5) repeats the update process until $\mathbf{W}$ converges. The computation cost of Equation (3.3) is $\mathcal{O}(p^2 n)$ time. Therefore, Equation (3.2) also requires $\mathcal{O}(p^2 n)$ time. Equation (3.2) can be modified to have $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ time as described in (Huang et al., 2012). However, in any case, the computation cost is still large because p^2 or pn is prohibitively large for a large data matrix.

3.3 Proposed Approach

This section presents our fast deterministic CUR. First, we provide an overview of our ideas in Section 3.3.1. Next, we provide their full descriptions in Sections 3.3.2, 3.3.3, and 3.3.4. We then describe our algorithm in Section 3.3.5. Finally, we introduce an extension of our algorithm for the simultaneous selection of both $\mathbf{R}$ and $\mathbf{C}$ in Section 3.3.6. The omitted proofs can be found in Section 3.7.

3.3.1 Ideas

To obtain the solution of CUR, coordinate descent requires a long processing time since the computation cost of Equation (3.2) is high, and the equation is performed for all the parameter vectors at every iteration until convergence.

To speed up coordinate descent, we safely skip the computations for the rows of $\mathbf{W}$ that must be updated to *zero* vectors during the optimization. Our approach can effectively reduce the computation cost by skipping unnecessary computations of Equation (3.2). To identify the unnecessary rows of $\mathbf{W}$, we approximately evaluate the necessary and sufficient conditions for the parameter vectors to be zero vectors.

In particular, we compute *upper* bounds of the condition scores instead of the exact scores. The computation of the exact score requires $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ time. On the other hand, because the computation of the upper bound requires only $\mathcal{O}(p)$ time, we can effectively reduce the processing time of the coordinate descent.

Another idea is to preferentially update the parameter vectors $\mathbf{W}_{(i)}$ that must be updated to *nonzero* vectors. Since these nonzero parameter vectors correspond to the columns $\mathbf{C}$, as explained in Section 3.2.1, we can expect the algorithm to effectively optimize the objective by intensively updating the parameter vectors. We utilize *lower* bounds of the condition scores for the parameter vectors to be updated to zero vectors to identify such parameter vectors. Because the additional computation cost is $\mathcal{O}(p)$ time, we can efficiently identify the parameter vectors that must be updated to nonzero vectors.

3.3.2 Approximation of Condition Score for Zero Parameter

This section introduces the key approximations of the condition scores of the necessary and sufficient conditions for the parameter vectors to be zero vectors: the upper and lower bounds of the scores. As described in Section 3.3.1, the upper and lower bounds are used to identify the parameter vectors that must be updated to zero vectors and nonzero vectors, respectively. First, we introduce the lemma of the necessary and sufficient condition for the zero parameter vector and its score as follows:

Lemma 3.1 (Necessary and Sufficient Condition for Zero Parameter) *Let $K_i := \|\boldsymbol{z}_i\|_2$ be the condition score for $\boldsymbol{W}_{(i)}$, where $\boldsymbol{z}_i$ is computed using Equation (3.3). Suppose that $\boldsymbol{W}_{(j)}$ is fixed for $i \neq j$. Then, we have $\boldsymbol{W}_{(i)} = \boldsymbol{0}$ if and only if $K_i \leq \lambda$.*

We can check whether $\mathbf{W}_{(i)}$ is a zero vector by using Lemma 3.1. However, $\mathbf{z}_i$ in Lemma 3.1 incurs a high computation cost: it requires $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ time as described in Section 3.2.2. As a result, the computation cost of the condition score K_i is also $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ time. To overcome this problem, we approximately compute the condition score. In particular, we evaluate two types of approximated scores instead of the exact score. These scores are defined as follows:

Definition 3.1 *Let $\overline{K}_i$ and $\underline{K}_i$ be approximated scores of the condition score K_i in Lemma 3.1. $\tilde{K}_i$ and $\tilde{\boldsymbol{W}}$ denote the condition score and the parameter matrix before entering the inner loop (lines 3–4 in Algorithm 3.1) of the coordinate descent, respectively. We define $\overline{K}_i$ and $\underline{K}_i$ as follows:*

$$\overline{K}_i = \tilde{K}_i + \|\Delta \boldsymbol{W}_{(i)}\|_2 + \|\boldsymbol{G}_{(i)}\|_2 \|\Delta \boldsymbol{W}\|_F, \tag{3.4}$$

and

$$\underline{K}_i = \tilde{K}_i - \|\Delta \boldsymbol{W}_{(i)}\|_2 - \|\boldsymbol{G}_{(i)}\|_2 \|\Delta \boldsymbol{W}\|_F, \tag{3.5}$$

where $\Delta \boldsymbol{W}_{(i)} = \boldsymbol{W}_{(i)} - \tilde{\boldsymbol{W}}_{(i)}$ and $\Delta \boldsymbol{W} = \boldsymbol{W} - \tilde{\boldsymbol{W}}$. $\boldsymbol{G}_{(i)} \in \mathbb{R}^{1\times p}$ is the i-th row vector of $\boldsymbol{G} := \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} \in \mathbb{R}^{p\times p}$.

Note that we can precompute $\tilde{K}_i$ and $\|\mathbf{G}_{(i)}\|_2$ before entering the inner loop and the outer loop, respectively. Although these approximated scores still require $\mathcal{O}(p^2)$ time, we introduce an efficient computation for them in Sections 3.3.3 and 3.3.4. The following lemma shows that $\overline{K}_i$ and $\underline{K}_i$ are upper and lower bounds of K_i, respectively:

Lemma 3.2 (Upper and Lower Bounds) *We have $\overline{K}_i \geq K_i$ and $\underline{K}_i \leq K_i$ for the approximated scores given in Definition 3.1.*

We use the upper and lower bounds as the approximations of the condition score. The error bounds of the approximations are given as follows:

Lemma 3.3 (Error Bound) *Let ϵ be defined as $2\|\Delta \boldsymbol{W}_{(i)}\|_2 + 2\|\boldsymbol{G}_{(i)}\|_2\|\Delta \boldsymbol{W}\|_F$. Then, we have $|\overline{K}_i - K_i| \leq \epsilon$ and $|\underline{K}_i - K_i| \leq \epsilon$ for the upper and lower bounds, respectively.*

3.3.3 Skipping Computations

This section introduces our first idea to skip the computations for the rows of $\mathbf{W}$ that must be updated to *zero* vectors during the optimization. Because Equation (3.2) requires a high computation cost, we expect the coordinate descent to reduce the processing time by skipping the computations. To identify $\mathbf{W}_{(i)}$ that must be updated to a zero vector, we utilize the upper bound of the condition score $\overline{K}_i$ in Definition 3.1. Specifically, we utilize the following property of the upper bound:

Lemma 3.4 (Rows with Zero Vectors) *When $\overline{K}_i \leq \lambda$ holds, we have $\boldsymbol{W}_{(i)} = \boldsymbol{0}$ for the i-th row vector.*

According to Lemma 3.4, we can identify the rows that must be updated to zero vectors by using the upper bounds $\overline{K}_i$. However, the computation cost of the upper bound is still high because Equation (3.4) requires $\mathcal{O}(p^2)$ time even if we precompute $\tilde{K}_i$ and $\|\mathbf{G}_{(i)}\|_2$ due to the computation of $\|\Delta\mathbf{W}\|_F$. Since the previous approaches require $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ times for Equation (3.2), the computation of the upper bound is not very efficient. Therefore, we introduce an efficient computation for the upper bound. In particular, we perform an online update for $\|\Delta\mathbf{W}\|_F$ in the upper bound when a parameter vector is updated as follows:

Lemma 3.5 (Online Update) *When $\boldsymbol{W}_{(j)}$ is updated to $\boldsymbol{W}'_{(j)}$, an upper bound $\overline{K}_{i\neq j}$ is computed as follows:*

$$\overline{K}_i = \tilde{K}_i + \|\Delta\boldsymbol{W}_{(i)}\|_2 + \delta\|\boldsymbol{G}_{(i)}\|_2, \tag{3.6}$$

where

$$\delta = \sqrt{\|\Delta\boldsymbol{W}\|_F^2 - \|\Delta\boldsymbol{W}_{(j)}\|_2^2 + \|\Delta\boldsymbol{W}'_{(j)}\|_2^2}. \tag{3.7}$$

The above online updating scheme[1] requires the following computation cost:

Lemma 3.6 (Computation Cost for Online Update) *Given precomputed $\tilde{K}_i$ and $\|\boldsymbol{G}_{(i)}\|_2$, the computation cost of Equation (3.6) is $\mathcal{O}(p)$ time when $\boldsymbol{W}_{(j)}$ is updated.*

Lemma 3.6 shows that our online updating scheme can compute the upper bound within $\mathcal{O}(p)$ time, which is lower than $\mathcal{O}(p^2)$ or $\mathcal{O}(pn)$ times for Equation (3.2) in the previous approaches. Therefore, we can efficiently identify the rows that must be updated to zero vectors, and skip the computation of Equation (3.2), which incurs a high computation cost.

Although the upper bound can be efficiently computed, the error bound of the upper bound ϵ in Lemma 3.3 is also important for reducing the processing time. This

[1]We note that $\|\Delta\mathbf{W}'_{(i)}\|_2$ is used instead of $\|\Delta\mathbf{W}_{(i)}\|_2$ in Equation (3.6) when $i = j$; however, this case would not be happen in our algorithm because the coordinate descent updates the parameter vectors in a cyclic order.

is because it is difficult to maintain the condition $\overline{K}_i \leq \lambda$ in Lemma 3.4 if ϵ is large. As a result, the number of skipped rows may be moderate. To increase the number of skipped rows, ϵ should be small. Fortunately, the error bounds of the upper and lower bounds have the following advantage:

Lemma 3.7 (Convergence of Error Bound) *If $\boldsymbol{W}$ reaches convergence by the coordinate descent, we have $\epsilon = 0$ for the error bound. Namely, the upper bound $\overline{K}_i$ and the lower bound $\underline{K}_i$ converge to the exact condition score K_i when $\boldsymbol{W}$ converges.*

Lemma 3.7 indicates that the upper bound matches the condition score when the parameter converges. Intuitively, the upper bound becomes increasingly tighter during the optimization as the bound depends on $\Delta\mathbf{W}_{(i)}$ and $\Delta\mathbf{W}$. Since ϵ becomes increasingly smaller during the optimization, the upper bounds can accurately identify the rows that must be updated to zero vectors as the optimization progresses. As a result, we can effectively skip the computations of the rows by using the upper bound.

3.3.4 Selective Update

This section presents our second idea to preferentially update $\mathbf{W}_{(i)}$ that must be a *nonzero* vector. Since the nonzero parameter vectors correspond to the columns $\mathbf{C}$, we can expect the coordinate descent to effectively optimize the objective by intensively updating the nonzero parameter vectors. Specifically, we first construct the set including rows that must be updated to nonzero vectors. Next, we perform coordinate descent on the set. We then update all the parameter vectors via the coordinate descent with the upper bounds until convergence.

To find $\mathbf{W}_{(i)}$ that must be updated to a nonzero vector, we utilize the lower bound of the condition score $\underline{K}_i$ in Definition 3.1. Similarly to the upper bound, the lower bound has the following property:

Lemma 3.8 (Rows with Nonzero Vectors) *When $\underline{K}_i > \lambda$ holds, we have $\boldsymbol{W}_{(i)} \neq \boldsymbol{0}$ for the i-th row vector.*

Lemma 3.8 shows that we can identify a row that must have a nonzero vector by using the lower bound $\underline{K}_i$. We define a subset of the rows on the basis of Lemma 3.8 as follows:

Definition 3.2 (Row Set $\mathbb{M}$) *We define the set $\mathbb{M}$ by using the lower bound $\underline{K}_i$ as follows:*

$$\mathbb{M} = \{i \in \{1, ..., p\} | \underline{K}_i > \lambda\}. \tag{3.8}$$

The set $\mathbb{M}$ has the following property:

Lemma 3.9 (Row Set $\mathbb{M}$) *The set $\mathbb{M}$ contains the rows that must be updated to nonzero vectors.*

As shown in Definition 3.2, the computation cost to construct the set $\mathbb{M}$ depends on the computation cost of the lower bound $\underline{K}_i$. Similarly to the computation of the upper bound, the lower bound requires $\mathcal{O}(p^2)$ time for the original computation of Equation (3.5) and $\mathcal{O}(p)$ time for the online updating scheme similar to Lemma 3.5. Since we must check the condition of Lemma 3.8 for each row to construct the set $\mathbb{M}$, we need $\mathcal{O}(p^2)$ time even if we use the online updating scheme. This is the motivation behind the lower bound computation using the upper bound. In particular, after the upper bound is updated by following Lemma 3.5, we compute the lower bound by utilizing the error bound of Lemma 3.3 as follows:

Lemma 3.10 (Computation using Upper Bound) *After the upper bound $\overline{K}_i$ is computed by using Equations (3.6) and (3.7), the lower bound $\underline{K}_i$ is computed as follows:*

$$\underline{K}_i = \overline{K}_i - 2\|\Delta \boldsymbol{W}_{(i)}\|_2 - 2\delta\|\boldsymbol{G}_{(i)}\|_2. \tag{3.9}$$

The computation cost of Equation (3.9) is as follows:

Lemma 3.11 (Computation Cost for Lower Bound) *After the upper bound is computed using Equations (3.6) and (3.7), Equation (3.9) requires $\mathcal{O}(1)$ time.*

Lemma 3.11 shows that the lower bound can be efficiently computed after the computation of the upper bound. This is because we can reuse the computed variables of the upper bounds for the computation of the lower bounds. Finally, we obtain the cost of constructing the set $\mathbb{M}$ as follows:

Lemma 3.12 (Computation Cost for Set $\mathbb{M}$) *We can construct the set $\mathbb{M}$ in Lemma 3.9 in $\mathcal{O}(p)$ time after the upper bounds are computed.*

The computation cost of $\mathcal{O}(p)$ time is significantly lower compared with the original computation based on Equation (3.5), which requires $\mathcal{O}(p^3)$ time to construct the set $\mathbb{M}$.

3.3.5 Algorithm

Algorithm 3.2 shows the pseudocode of our algorithm, namely Fast Deterministic CUR, which utilizes the above-mentioned definitions and lemmas. The algorithm utilizes two ideas as described in the previous sections: i) it safely skips the computations for the rows of $\mathbf{W}$ that must be updated to zero vectors by using the upper bounds $\overline{K}_i$, and ii) it preferentially updates the parameter vectors corresponding to the row set $\mathbb{M}$, which must be updated to nonzero vectors. Since the original deterministic CUR has the regularization constant λ as described in problem (3.1), we tune the regularization constant by using the sequential rule (Ghaoui et al., 2012) with a warm start (Friedman et al., 2007), which is a standard approach for optimization with sparsity-inducing norms (Wang and Ye, 2014; Ndiaye et al., 2017; Ida et al., 2019). Specifically, it sequentially tunes the regularization constant λ with respect to the sequence $(\lambda_q)_{q=0}^{Q-1}$, where $\lambda_0 > \lambda_1 > ... > \lambda_{Q-1}$. The parameter matrices $\mathbf{W}$ are sequentially optimized for each regularization constant by using the coordinate descent, and the initial parameter matrix of the current λ_q is the result of the previous λ_{q-1}.

In Algorithm 3.2, we first precompute $\|\mathbf{G}_{(i)}\|_2$ for computing the upper bounds and lower bounds (lines 2–3). We next enter the loop of the sequential rule (lines 4–25). In the loop, we construct the set $\mathbb{M}$ by using the lower bounds of the condition scores (lines 5–9). If the lower bound $\underline{K}_i$ is larger than λ_q, we add the index of the row to $\mathbb{M}$ (lines 8–9). It should be noted that when $\underline{K}_i$ is computed, we use the $\tilde{K}_i$, $\Delta\mathbf{W}_{(i)}$, and δ used in the last computation of the upper bound in the previous loop of the sequential rule. If $q = 0$, we compute the lower bounds by the original definition of Equation (3.5). Another strategy for computing the lower bounds in the case of $q = 0$ is to first run lines 15–24 once and compute the lower bounds by using Equation (3.9). After constructing the set $\mathbb{M}$, we perform coordinate descent on the set (lines 10–13). We then enter the loop of the coordinate descent with the upper bounds of the condition scores (lines 14–25). We set $\tilde{\mathbf{W}}$ in line 15 and compute $\tilde{K}_i$ (lines 16–17). The computation results are used to compute the upper bounds $\overline{K}_i$

(line 19). If the upper bound is less than or equal to λ_q, $\mathbf{W}_{(i)}$ turns to be $\mathbf{0}$ (lines 20–21). In other cases, $\mathbf{W}_{(i)}$ is updated as usual (lines 22–23). Subsequently, we update δ by following the online updating scheme of Lemma 3.5 (line 17).

The computation cost of Algorithm 3.2 is given as follows:

Theorem 3.1 (Computation Cost) *Let t_{u} be the total number of outer loops for the coordinate descent with the upper bounds. Suppose that S is the ratio of updates to the total number of inner loops, which is un-skipped by the upper bounds. If t_{m} is the total number of inner loops for the set $\mathbb{M}$, Algorithm 3.2 requires $\mathcal{O}(p\{n(t_{\mathrm{m}} + pt_{\mathrm{u}}S) + Q\})$ time.*

We expect the algorithm to reduce t_{u} by preferentially updating the parameter vectors on the basis of the set $\mathbb{M}$ in Lemma 3.9. In addition, S would be small when the algorithm skips a large number of updates by utilizing the upper bounds in Lemma 3.4. As a result, the total computation cost would be effectively reduced.

In terms of the optimization result, Algorithm 3.2 has the following property:

Theorem 3.2 (Optimization Result) *Suppose that Algorithm 3.2 has the same regularization constants $\lambda_q > 0$ as those of the original algorithm, and the coordinate descent converges to a global optimal solution. Then, Algorithm 3.2 converges to the same objective values as those of the original algorithm.*

The aforementioned theorem suggests that our algorithm achieves the same accuracy as that of the original algorithm under the assumption of convergence to a global optimal solution. Therefore, we expect Algorithm 3.2 to speed up the deterministic CUR without degrading the accuracy. This property corresponds to Theorem 2.2 in Section 2.3.5. Thus, our algorithm returns the same parameters and objective value as those of the original algorithm if we assume that the order of the updates in our algorithms is the same as the original algorithm.

Algorithm 3.2 Fast Deterministic CUR

$\mathbb{A} = \{1, ..., p\}$, $\mathbf{W} \leftarrow \mathbf{0}$, $\tilde{\mathbf{W}} \leftarrow \mathbf{0}$, $\mathbf{G} \leftarrow \mathbf{X}^{\mathrm{T}}\mathbf{X}$
for each $i \in \mathbb{A}$ **do**
    compute $\|\mathbf{G}_{(i)}\|_2$;
for $q = 0$ to $Q - 1$ **do**
    $\mathbb{M} = \emptyset$;
    for each $i \in \mathbb{A}$ **do**
        compute the lower bound $\underline{K}_i$ by Equation (3.9);
        if $\underline{K}_i > \lambda_q$ **then**
            add i to $\mathbb{M}$;
    repeat
        for each $i \in \mathbb{M}$ **do**
            update $\mathbf{W}_{(i)}$ by Equation (3.2);
    until $\mathbf{W}$ converges
    repeat
        $\tilde{\mathbf{W}} \leftarrow \mathbf{W}$;
        for each $i \in \mathbb{A}$ **do**
            compute $\tilde{K}_i$;
        for each $i \in \mathbb{A}$ **do**
            compute the upper bound $\overline{K}_i$ by Equation (3.6);
            if $\overline{K}_i \leq \lambda_q$ **then**
                $\mathbf{W}_{(i)} \leftarrow \mathbf{0}$;
            else
                update $\mathbf{W}_{(i)}$ by Equation (3.2);
            update δ by Equation (3.7);
    until $\mathbf{W}$ converges

3.3.6 Extension

We propose Algorithm 3.2 on the basis of problem (3.1) by following (Bien et al., 2010). Although it only selects the columns $\mathbf{C}$ as described in Section 3.2, Mairal et al. (2011) naturally extended the problem to achieve simultaneous selection of both rows $\mathbf{R}$ and columns $\mathbf{C}$ by using the row-wise and column-wise regularization terms. Although these regularization terms overlap, we can handle the terms by using the overlapping norm (Jacob et al., 2009). Specifically, we define the following optimization problem for the simultaneous selection by combining the extension of (Mairal et al., 2011) and the overlapping norm (Jacob et al., 2009):

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times n}} (\tfrac{1}{2}\|\mathbf{X}-\mathbf{X}\mathbf{W}\mathbf{X}\|_F^2+\lambda_r\textstyle\sum_{i=1}^{p}\|\mathbf{V}_{(i)}\|_2+\lambda_c\sum_{j=1}^{n}\|\mathbf{H}^{(j)}\|_2), \tag{3.10}$$

where $\mathbf{V}\in\mathbb{R}^{p\times n}$ and $\mathbf{H}\in\mathbb{R}^{p\times n}$ are latent variables: the parameter matrix is decomposed into a sum of the latent variables as $\mathbf{W}=\mathbf{V}+\mathbf{H}$; $\sum_{i=1}^{p}\|\mathbf{V}_{(i)}\|_2$ and $\sum_{j=1}^{n}\|\mathbf{H}^{(j)}\|_2$ are the overlapping norms, which correspond to row-wise and column-wise regularization terms, respectively; and $\lambda_r, \lambda_c \geq 0$ are regularization constants for the norms.

To solve problem (3.10), we perform row-wise and column-wise coordinate descents. Therefore, we have two types of condition scores for the parameters to be zeros as follows:

Lemma 3.13 (Necessary and Sufficient Condition for Zero Parameter) *Let* $R_i := \|\boldsymbol{X}^{(i)\mathrm{T}}\{\boldsymbol{X}-(\boldsymbol{X}\boldsymbol{W}-\boldsymbol{X}^{(i)}\boldsymbol{V}_{(i)})\boldsymbol{X}\}\boldsymbol{X}^{\mathrm{T}}\|_2$ *and* $C_j := \|\boldsymbol{X}^{\mathrm{T}}\{\boldsymbol{X}-\boldsymbol{X}(\boldsymbol{W}\boldsymbol{X}-\boldsymbol{H}^{(j)}\boldsymbol{X}_{(j)})\}\boldsymbol{X}_{(j)}^{\mathrm{T}}\|_2$ *be the condition scores for the parameters to be zeros for* $\boldsymbol{V}_{(i)}$ *and* $\boldsymbol{H}^{(j)}$*, respectively. Then, we obtain* $\boldsymbol{V}_{(i)}=\boldsymbol{0}$ *if and only if* $R_i \leq \lambda_r$*, and* $\boldsymbol{H}^{(j)}=\boldsymbol{0}$ *if and only if* $C_j \leq \lambda_c$*.*

We note that the columns $\mathbf{C}$ and the rows $\mathbf{R}$ are selected on the basis of the indices corresponding to the nonzero rows in $\mathbf{V}$ and nonzero columns in $\mathbf{H}$, respectively. Namely, if $\mathcal{I}\subseteq\{1,...,p\}$ and $\mathcal{J}\subseteq\{1,...,n\}$ are the indices corresponding to the nonzero rows in $\mathbf{V}$ and nonzero columns in $\mathbf{H}$ respectively, we obtain $\mathbf{X}^{\mathcal{I}}$ and $\mathbf{X}_{\mathcal{J}}$ as columns $\mathbf{C}$ and rows $\mathbf{R}$, respectively. If we need $\mathbf{U}$ after selecting $\mathbf{X}^{\mathcal{I}}$ and $\mathbf{X}_{\mathcal{J}}$, $\mathbf{U}$ is given by computing $(\mathbf{X}^{\mathcal{I}})^{+}\mathbf{X}(\mathbf{X}_{\mathcal{J}})^{+}$ where $\mathbf{A}^{+}$ represents the Moore-Penrose generalized inverse of matrix $\mathbf{A}$. Then, we can compute the approximated scores for the condition scores as follows:

Definition 3.3 *We define* $\overline{R}_i$ *and* $\underline{R}_i$ *be approximated scores of the condition score* R_i*.* $\tilde{R}_i$ *denotes the condition score before entering the inner loop of the coordinate descent. Then,* $\overline{R}_i$ *and* $\underline{R}_i$ *are respectively computed as* $\overline{R}_i = \tilde{R}_i + \rho_i$ *and* $\underline{R}_i = \tilde{R}_i - \rho_i$*, where* $\rho_i := \boldsymbol{G}_{(i)}^{(i)} \|\Delta \boldsymbol{V}_{(i)}\|_2 \|\boldsymbol{F}\|_F + \|\boldsymbol{G}_{(i)}\|_2 \|\Delta \boldsymbol{W}\|_F \|\boldsymbol{F}\|_F$*, and* $\boldsymbol{F} := \boldsymbol{X}\boldsymbol{X}^{\mathrm{T}} \in \mathbb{R}^{n \times n}$*. Similarly, the approximated scores of* $\overline{C}_j$ *and* $\underline{C}_j$ *for the condition score* C_j *are respectively computed as* $\overline{C}_j = \tilde{C}_j + \sigma_j$ *and* $\underline{C}_j = \tilde{C}_j - \sigma_j$*, where* $\sigma_j := \|\boldsymbol{G}\|_F \|\Delta \boldsymbol{H}^{(j)}\|_2 \boldsymbol{F}_{(j)}^{(j)} + \|\boldsymbol{G}\|_F \|\Delta \boldsymbol{W}\|_F \|\boldsymbol{F}^{(j)}\|_2$.

Similarly to the proof of Lemma 3.2, we can show that $\overline{R}_i$ and $\underline{R}_i$ are the upper and lower bounds for R_i from Lemma 3.13, respectively. In addition, $\overline{C}_j$ and $\underline{C}_j$ are also the upper and lower bounds for C_j, respectively. We note that the assumption of $\|\mathbf{X}^{(i)}\|_2 = 1$ in Lemma 3.2 is not required for Definition 3.3. These bounds can realize our two ideas as described in the chapter: i) safely skipping the computations for rows and columns by using the upper bounds, and ii) prioritizing the update order by using the lower bounds.

3.4 Related Work

The deterministic CUR is a convex optimization problem with sparsity-inducing norms as shown in problem (3.1). As these norms correspond to the group-level regularizations used in group Lasso (Yuan and Lin, 2006), we can use several techniques for group Lasso to speed up CUR.

To efficiently solve group Lasso, screening rules (Tibshirani et al., 2012; Wang et al., 2013; Bonnefoy et al., 2015; Ndiaye et al., 2017) are popular methods, which eliminate several parameters before achieving optimization. Since they reduce the size of the problem, we can expect the optimization algorithm to reduce the processing time. Dual Polytope Projections (DPP) is a screening rule that utilizes the geometric property of the dual solution (Wang et al., 2013). Unfortunately, DPP may eliminate parameters incorrectly because it theoretically requires the exact solution of the previous λ in the sequential rule, which is not practically available in the iterative optimization algorithm. Bonnefoy et al. (2015) proposed a dynamic method of screening: it eliminates the parameters, not only before the optimization but also during the optimization. As a result, it can be expected to eliminate a large number of parameters. However, if the number of eliminated parameters is small,

the processing time can increase due to the overhead of the screening process (Ida et al., 2019).

Sequential strong rule (SSR) (Tibshirani et al., 2012) is a heuristic strategy for screening, which approximately eliminates the parameters. Although it can eliminate parameters incorrectly, this can be avoided by checking the Karush–Kuhn–Tucker (KKT) condition after the optimization. In spite of the fact that SSR is a relatively old method, it still achieves the state-of-the-art results compared with the recent screening methods (Ndiaye et al., 2017).

Although our idea of selective update using lower bounds in Section 3.3.4 can be seen as a screening, we utilize another idea of skipping computations using upper bounds in Section 3.3.3. As a general problem, screenings could not speed up the optimization when the number of eliminated features is small (Johnson and Guestrin, 2016; 2017; Ida et al., 2019). On the other hand, our method can effectively skip updates by using upper bounds during the optimization even if lower bounds cannot eliminate so many features because our bounds become increasingly tighter during the optimization as shown in Lemma 3.7. In addition, our screening using lower bounds is efficient because it only requires $\mathcal{O}(p)$ time as shown in Lemma 3.12.

All the aforementioned methods including our method can further improve the efficiency by utilizing a warm start strategy (Friedman et al., 2007; Ndiaye et al., 2017). The warm start uses the solution of the previous λ as the initial parameters of the current λ in the sequential rule. As a result, it empirically speeds up the optimization algorithm.

3.5 Experiments

We evaluated the processing times and values of the objectives. We compared our method with the original deterministic CUR (*origin*) (Bien et al., 2010), the sequential strong rule (*SSR*) (Tibshirani et al., 2012), and the sequential strong rule with warm start (*SSR+WS*) (Ndiaye et al., 2017). We tuned λ for all the approaches based on the sequential rule by following the methods in (Tibshirani et al., 2012; Wang and Ye, 2014; Ndiaye et al., 2017; Ida et al., 2019). The search space was a non-increasing sequence of Q parameters $(\lambda_q)_{q=0}^{Q-1}$ defined as $\lambda_q = \lambda_{max} 10^{-\gamma q/Q-1}$. λ_{max} is the smallest λ for which all the parameters are zeros at the optimal solutions and it was

computed by following (Tibshirani et al., 2012). We used $\gamma = 4$ and $Q = 100$ (Wang and Ye, 2014; Ndiaye et al., 2017; Ida et al., 2019). We stopped the algorithm for each λ_q when the relative tolerance of the parameter matrix dropped below 10^{-5} for all the approaches (Johnson and Guestrin, 2016; 2017; Ida et al., 2019). We stopped the sequential rule when all the parameters were nonzeros since our purpose is to select the subset of the columns. We conducted the experiments on five datasets from the LIBSVM[2] (Chang and Lin, 2011) and OpenML[3] (Vanschoren et al., 2013) websites, namely colon-cancer, Bioresponse, QSAR-TID-52, Madelon, and Slashdot, and the sizes of the data matrices were 62×2000, 3751×1776, 877×1024, 2000×500, and 3782×1079, respectively. Each experiment was conducted with one CPU core and 264 GB of main memory on a 2.20 GHz Intel Xeon server running Linux.

3.5.1 Processing Time

We evaluated the processing times of the sequential rules for each method. Table 3.1 shows the wall clock times for the five datasets. Note that the processing times include precomputation times for a fair comparison. Our method was faster than the previous methods for all the datasets. It reduced the processing time by up to 34.64% compared to the original method. Although SSR+WS was competitive in comparison to our method on the Madelon dataset, our method was faster than SSR+WS on the other datasets. This is because the speed-up of SSR+WS is moderate due to the overheads of checking the KKT conditions when the number of eliminated parameters is small. Unfortunately, SSR and SSR+WS could not finish the computations within a month on the colon-cancer and Bioresponse datasets due to the overheads. On the other hand, our method quickly extracts the parameter vectors that must be updated to nonzero vectors at $\mathcal{O}(p)$ time. In addition, it efficiently skips the parameter vector that must be updated to a zero vector at $\mathcal{O}(p)$. Thanks to the small overheads, our method reduces the processing times even on datasets that are not suitable for SSR and SSR+WS.

[2] `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`
[3] `https://www.openml.org/`

Table 3.1: Wall clock times on each dataset. We omit some computation results, which could not finish within a month.

Dataset	Wall clock time (s)				Reduction ratio (%)
	origin	SSR	SSR+WS	ours	(ours/origin)
colon-cancer	1.384×10^6	–	–	$\mathbf{4.793 \times 10^5}$	**34.64**
Bioresponse	1.461×10^6	–	–	$\mathbf{9.908 \times 10^5}$	**67.80**
QSAR-TID-52	9.628×10^3	1.301×10^4	8.875×10^3	$\mathbf{6.323 \times 10^3}$	**65.67**
Madelon	2.555×10^3	1.194×10^3	1.008×10^3	$\mathbf{1.006 \times 10^3}$	**39.36**
Slashdot	1.827×10^3	2.122×10^3	1.636×10^3	$\mathbf{8.592 \times 10^2}$	**47.04**

Table 3.2: Numbers of updates of Eq. (3.2). Our method effectively reduces the number of bottleneck computations.

Dataset	# of updates of Eq. (3.2)	
	origin	ours
colon-cancer	8.344×10^7	$\mathbf{4.181 \times 10^7}$
Bioresponse	9.048×10^7	$\mathbf{7.655 \times 10^7}$
QSAR-TID-52	1.991×10^6	$\mathbf{1.131 \times 10^6}$
Madelon	1.847×10^6	$\mathbf{2.162 \times 10^5}$
Slashdot	6.571×10^5	$\mathbf{1.898 \times 10^5}$

Table 3.2 shows the numbers of updates of Equation (3.2) in the original method and our method on each dataset. The results suggest that our approximations of the upper and lower bounds effectively reduced the number of updates by skipping unnecessary updates and preferentially updating the parameters. Since Equation (3.2) is the main bottleneck as described in Section 3.2.2, our method could effectively reduce the processing time as shown in Table 3.1.

Figure 3.1 shows the mean and standard deviation for approximation errors of the upper and lower bounds during the optimization (lines 14–25 in Algorithm 3.2). The result was obtained with $\lambda_q = 1.2$ on the Slashdot dataset; the ratio of zero parameters was 80% in this setting. Similar results were obtained for the other settings. This result suggests that the error bounds become increasingly smaller during the optimization; it supports our theoretical result of Lemma 3.7. Namely, the upper and lower bounds become tight during the optimization. Thanks to the small approximation errors, our method effectively identifies the parameter vectors that must be updated to nonzero and zero vectors by using the lower and upper bounds, respectively.

We also investigated the relationship between the processing time and the size of the data matrix. We used the gene expression data from (Ramaswamy et al., 2002), which is a data matrix of 190×16063. We randomly sampled 100, 500, 1000, 5000, and 10000 columns from the data matrix, and evaluated the processing times to select 1% of all the columns from these five data matrices on two CPU cores. As shown in Figure 3.2, our method was faster than the existing methods for all

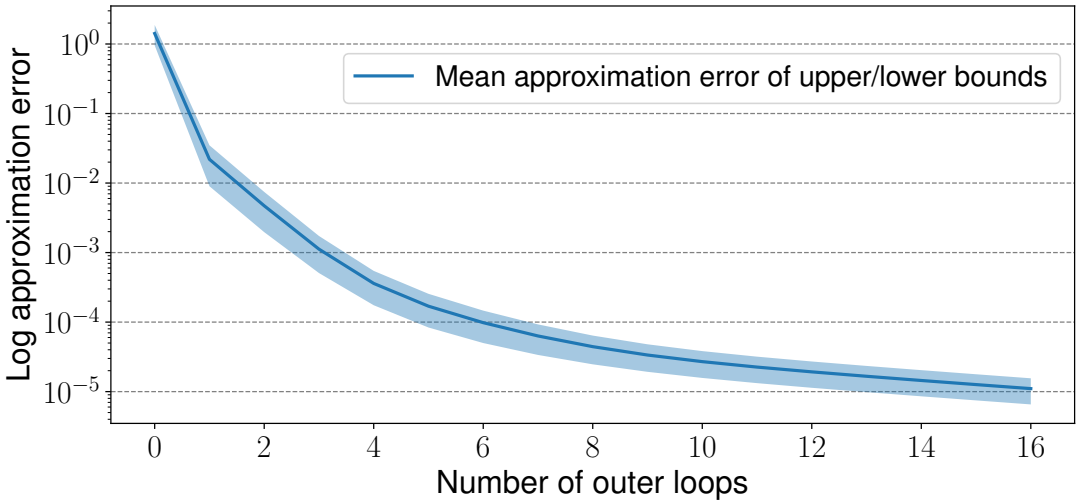


Figure 3.1: Mean and standard deviation for approximation errors of the upper and lower bounds during optimization. Our bounds become tight during the optimization.

sizes. Specifically, for 1000 columns, our method was 10× and 4× faster than the original method and SSR+WS, respectively. We omit the results of the comparison methods for 10000 columns because they could not finish the computations within a month. The results show that our method achieves higher efficiency than the existing methods.

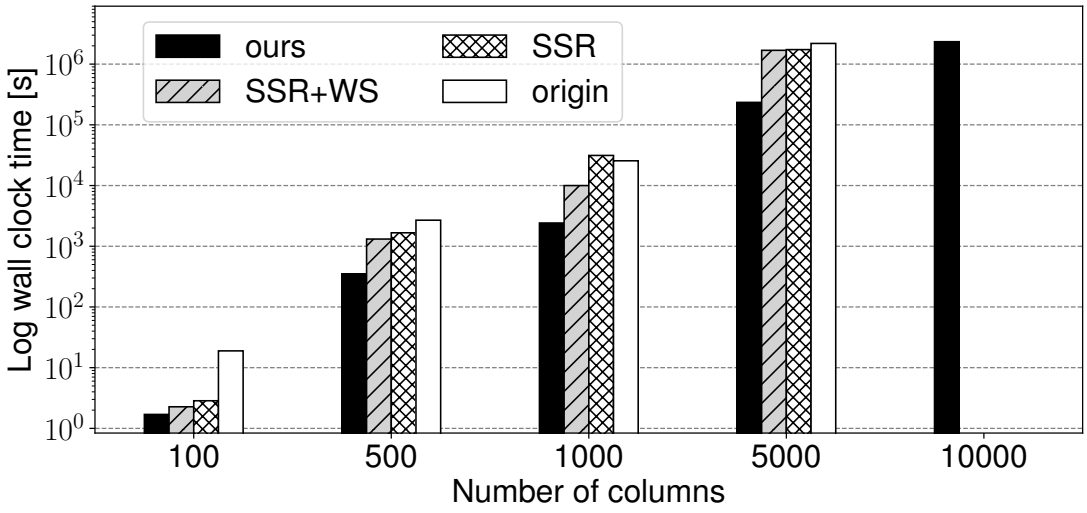


Figure 3.2: Log wall clock time vs. number of columns with the gene expression data. Our method is up to 10× and 4× faster than the original method and SSR+WS, respectively. We omit some computation results which could not finish within a month.

Table 3.3: Values of objectives. Our method converges to the same objective as that of the original method.

Dataset	Objective	
	origin	ours
colon-cancer	**1.336**	**1.336**
Bioresponse	$\mathbf{9.683 \times 10^1}$	$\mathbf{9.683 \times 10^1}$
QSAR-TID-52	$\mathbf{5.727 \times 10^2}$	$\mathbf{5.727 \times 10^2}$
Madelon	$\mathbf{9.390 \times 10^2}$	$\mathbf{9.390 \times 10^2}$
Slashdot	$\mathbf{1.837 \times 10^3}$	$\mathbf{1.837 \times 10^3}$

3.5.2 Accuracy

We evaluated the value of the objective function to confirm the effectiveness of our method. Table 3.3 shows the results: they are final values of the objective functions in the sequential rules. The values of the objectives of our method are the same as those of the original method. This is because our method is guaranteed to yield the same value of the objective function as that of the original method, as described in Theorem 3.2. We note that SSR and SSR+WS also achieved the same objectives as those of the original method for QSAR-TID-52, Madelon, and Slashdot datasets because they are safe screening methods. However, they could not finish the computations within a month on the other datasets as shown in Table 3.1. Table 3.3 shows that our method can maintain the accuracy while speeding up CUR.

3.6 Summary

We proposed a fast deterministic CUR matrix decomposition. The main bottleneck of the original method is the coordinate descent, which requires a large number of parameter updates. Our method utilizes two ideas to tackle this problem: i) it safely skips the updates by identifying the parameters that must be updated to *zeros*, and ii) it preferentially updates the parameters that must be updated to *nonzeros*. The key is to approximately evaluate the necessary and sufficient conditions for the parameter vectors to be zero vectors by using the upper and lower bounds of the condition scores. In addition, it provably guarantees the same results as those of

the original method. The experimental results showed that our method is up to 10× faster than the original method, and up to 4× faster than the state-of-the-art method without requiring any additional hyperparameters or incurring any loss of accuracy.

Although we handle only matrix decomposition in this chapter, our method can be extended to tensor decomposition. Furthermore, our ideas can be generalized for various types of structured data such as graph, tree, and heterogeneous data. These future works will enable a wide range of applications to be implemented more efficiently.

3.7 Proofs

Proof of Lemma 3.1

Proof *For the objective in problem (3.1),* $\boldsymbol{W}_{(i)} = \boldsymbol{0}$ *holds if and only if the following condition holds from (Sra et al., 2012):*

$$-\boldsymbol{z}_i + \lambda \boldsymbol{v}_i = \boldsymbol{0}, \tag{3.11}$$

where $\boldsymbol{v}_i$ *is an element of the subdifferential of* $\| \boldsymbol{W}_{(i)} \|_2$*. The subdifferential for the* l_2 *norm is represented as* $\partial \| \boldsymbol{W}_{(i)} \|_2 = \{ \boldsymbol{v}_i \in \mathbb{R}^{1 \times p} | \| \boldsymbol{v}_i \|_2 \leq 1 \}$ *if* $\boldsymbol{W}_{(i)} = \boldsymbol{0}$ *(Sra et al., 2012). Therefore, we obtain the condition* $K_i \leq \lambda$ *in Lemma 3.1 by using Equation (3.11) and the condition* $\| \boldsymbol{v}_i \|_2 \leq 1$*.* □

We note that Lemma 3.1 is a known result (see Hastie et al. (2015); Yuan and Lin (2006); Simon et al. (2013); Friedman et al. (2010); Sra et al. (2012) in detail).

Proof of Lemma 3.2

Proof *From Equation (3.3) and* $\| \boldsymbol{X}^{(i)} \|_2 = 1$*, we obtain*

$$\boldsymbol{z}_i = \boldsymbol{G}_{(i)} - \boldsymbol{G}_{(i)} \boldsymbol{W} + \boldsymbol{W}_{(i)}. \tag{3.12}$$

If $\tilde{\boldsymbol{z}}_i := \boldsymbol{G}_{(i)} - \boldsymbol{G}_{(i)}\tilde{\boldsymbol{W}} + \tilde{\boldsymbol{W}}_{(i)}$ *is* $\boldsymbol{z}_i$ *before entering the inner loop of the coordinate descent, Equation (3.12) is transformed into the following form:*

$$\begin{aligned}\boldsymbol{z}_i &= \boldsymbol{G}_{(i)} - \boldsymbol{G}_{(i)}\tilde{\boldsymbol{W}} + \tilde{\boldsymbol{W}}_{(i)} - \boldsymbol{G}_{(i)}\Delta\boldsymbol{W} + \Delta\boldsymbol{W}_{(i)} \\ &= \tilde{\boldsymbol{z}}_i - \boldsymbol{G}_{(i)}\Delta\boldsymbol{W} + \Delta\boldsymbol{W}_{(i)}. \end{aligned} \tag{3.13}$$

From the aforementioned equation and the triangle equality, we obtain the following inequality:

$$\|\boldsymbol{z}_i\|_2 \leq \|\tilde{\boldsymbol{z}}_i\|_2 + \|\Delta\boldsymbol{W}_{(i)}\|_2 + \|\boldsymbol{G}_{(i)}\Delta\boldsymbol{W}\|_2. \tag{3.14}$$

For the term $\|\boldsymbol{G}_{(i)}\Delta\boldsymbol{W}\|_2$*, we obtain the following inequality by using the Cauchy–Schwarz inequality:*

$$\|\boldsymbol{G}_{(i)}\Delta\boldsymbol{W}\|_2 \leq \|\boldsymbol{G}_{(i)}\|_2\|\Delta\boldsymbol{W}\|_F. \tag{3.15}$$

From Equations (3.14) and (3.15), we obtain the following upper bound in the lemma:

$$K_i \leq \tilde{K}_i + \|\Delta\boldsymbol{W}_{(i)}\|_2 + \|\boldsymbol{G}_{(i)}\|_2\|\Delta\boldsymbol{W}\|_F = \overline{K}_i. \tag{3.16}$$

We obtain $\|\boldsymbol{z}_i\|_2 \geq \|\tilde{\boldsymbol{z}}_i\|_2 - \|\Delta\boldsymbol{W}_{(i)}\|_2 - \|\boldsymbol{G}_{(i)}\Delta\boldsymbol{W}\|_2$ *for the lower bound by using Equation (3.13) and the triangle inequality, similarly to that in the case of the upper bound. From the inequality and Equation (3.15), we obtain the lower bound in the lemma:*

$$K_i \geq \tilde{K}_i - \|\Delta\boldsymbol{W}_{(i)}\|_2 - \|\boldsymbol{G}_{(i)}\|_2\|\Delta\boldsymbol{W}\|_F = \underline{K}_i, \tag{3.17}$$

which completes the proof. □

Proof of Lemma 3.3

Proof *From Lemma 3.2, we have* $\underline{K}_i \leq K_i \leq \overline{K}_i$*. Therefore, the error bound of the upper bound is* $|\overline{K}_i - K_i| \leq |\overline{K}_i - \underline{K}_i| = 2\|\Delta\boldsymbol{W}_{(i)}\|_2 + 2\|\boldsymbol{G}_{(i)}\|_2\|\Delta\boldsymbol{W}\|_F = \epsilon$*. Similarly to that for the upper bound, we obtain* $|\underline{K}_i - K_i| \leq |\underline{K}_i - \overline{K}_i| = \epsilon$ *for the lower bound.*

□

Proof of Lemma 3.4

Proof *When $\overline{K}_i \leq \lambda$ holds, we have $K_i \leq \overline{K}_i \leq \lambda$ from Lemma 3.2. Therefore, we have $\boldsymbol{W}_{(i)} = \boldsymbol{0}$ from Lemma 3.1 as $K_i \leq \lambda$ holds.* □

Proof of Lemma 3.5

Proof *For the term $\|\boldsymbol{G}_{(i)}\|_2 \|\Delta \boldsymbol{W}\|_F$ in Equation (3.4), since we have $\|\Delta \boldsymbol{W}\|_F^2 = \|[\|\Delta \boldsymbol{W}^{(1)}\|_2, ..., \|\Delta \boldsymbol{W}^{(p)}\|_2]\|_2^2 = \|\Delta \boldsymbol{W}^{(1)}\|_2^2 + ... + \|\Delta \boldsymbol{W}^{(p)}\|_2^2$, we can update $\|\Delta \boldsymbol{W}\|_F^2$ by using the following equation:*

$$\|\Delta \boldsymbol{W}\|_F^2 - \|\Delta \boldsymbol{W}_{(j)}\|_2^2 + \|\Delta \boldsymbol{W}'_{(j)}\|_2^2 = \delta^2 \tag{3.18}$$

Therefore, we obtain Equation (3.6) by using δ instead of $\|\Delta \boldsymbol{W}\|_F$ in Equation (3.4). □

Proof of Lemma 3.6

Proof *We can precompute $\tilde{K}_i$ and $\|\boldsymbol{G}_{(i)}\|_2$ before entering the inner loop and outer loop, respectively. In addition, we have $\|\Delta \boldsymbol{W}_{(i)}\|_2$ and $\|\Delta \boldsymbol{W}\|_F$ as scalars. Thus, we obtain the terms $\tilde{K}_i$, $\|\boldsymbol{G}_{(i)}\|_2$, $\|\Delta \boldsymbol{W}_{(i)}\|_2$, and $\|\Delta \boldsymbol{W}\|_F$ at $\mathcal{O}(1)$ times. When $\Delta \boldsymbol{W}_{(j)}$ is updated to $\Delta \boldsymbol{W}'_{(j)}$, the computation of $\|\Delta \boldsymbol{W}'_{(j)}\|_2$ in Equation (3.7) requires $\mathcal{O}(p)$ time. Therefore, the total computation cost of Equation (3.6) is $\mathcal{O}(p)$ time.* □

Proof of Lemma 3.7

Proof *If $\boldsymbol{W}$ converges, we have $\Delta \boldsymbol{W}_{(i)} = \boldsymbol{0}$ and $\Delta \boldsymbol{W} = \boldsymbol{0}$. Since the error bound ϵ is $2\|\Delta \boldsymbol{W}_{(i)}\|_2 + 2\|\boldsymbol{G}_{(i)}\|_2 \|\Delta \boldsymbol{W}\|_F$ in Lemma 3.3, we obtain $\epsilon = 0$. In addition, because we have $\tilde{K}_i = K_i$ when $\boldsymbol{W}$ converges, the upper bound $\overline{K}_i$ and the lower bound $\underline{K}_i$ converge to the condition score K_i from Equations (3.4) and (3.5).* □

Proof of Lemma 3.8

Proof *When $\underline{K}_i > \lambda$ holds, we have $K_i \geq \underline{K}_i > \lambda$ from Lemma 3.2. Therefore, since $K_i > \lambda$ holds, we have $\boldsymbol{W}_{(i)} \neq \boldsymbol{0}$ from Lemma 3.1.* □

Proof of Lemma 3.9

Proof *Lemma 3.9 holds since the set* $\mathbb{M}$ *includes indices of rows that must be updated to nonzero vectors from Lemma 3.8.* □

Proof of Lemma 3.10

Proof *Since* $\overline{K}_i \geq \underline{K}_i$ *from Lemma 3.2, the lower bound is computed as* $\underline{K}_i = \overline{K}_i - \epsilon$ *from the proof of Lemma 3.3. Since we have* $\epsilon = |\overline{K}_i - \underline{K}_i| = 2\|\Delta \boldsymbol{W}_{(i)}\|_2 + 2\delta\|\boldsymbol{G}_{(i)}\|_2$ *after* $\boldsymbol{W}_{(j)}$ *is updated to* $\boldsymbol{W}'_{(j)}$*, we obtain Equation (3.9).* □

Proof of Lemma 3.11

Proof *The terms* $\tilde{K}_i$*,* $\|\Delta \boldsymbol{W}_{(i)}\|_2$*,* $\delta = \|\Delta \boldsymbol{W}\|_F$*, and* $\|\boldsymbol{G}_{(i)}\|_2$ *in Equation (3.5) have already been computed in Equations (3.6) and (3.7). Since we obtain these terms at* $\mathcal{O}(1)$ *times, the computation cost of Equation (3.9) is* $\mathcal{O}(1)$ *time.* □

Proof of Lemma 3.12

Proof *Lemma 3.12 holds from Lemma 3.11 since the computation of the construction for the set* $\mathbb{M}$ *checks* $\underline{K}_i$*, which requires* $\mathcal{O}(1)$ *time, for* p *rows.* □

Proof of Lemma 3.13

Proof *Suppose that* $L(\boldsymbol{W}) := \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}\boldsymbol{X}\|_F^2$ *in problem (3.10). By following the property of the overlapping norm (Jacob et al., 2009),* $\boldsymbol{W}$ *is an optimal solution of problem (3.10) if and only if the following conditions hold for any rows and columns: (i)* $\boldsymbol{W}$ *can be decomposed as* $\boldsymbol{W} = \boldsymbol{V} + \boldsymbol{H}$*, (ii) if* $\boldsymbol{V}_{(i)} = \boldsymbol{0}$ *then* $\|\nabla_i L(\boldsymbol{W})\|_2 \leq \lambda_r$*, (iii) if* $\boldsymbol{V}_{(i)} \neq \boldsymbol{0}$ *then* $\nabla_i L(\boldsymbol{W}) = -\lambda_r \boldsymbol{V}_{(i)} / \|\boldsymbol{V}_{(i)}\|_2$*, (iv) if* $\boldsymbol{H}^{(j)} = \boldsymbol{0}$ *then* $\|\nabla^j L(\boldsymbol{W})\|_2 \leq \lambda_c$*, and (v) if* $\boldsymbol{H}^{(j)} \neq \boldsymbol{0}$ *then* $\nabla^j L(\boldsymbol{W}) = -\lambda_c \boldsymbol{H}^{(j)} / \|\boldsymbol{H}^{(j)}\|_2$*, where* $\nabla_i L(\cdot)$ *and* $\nabla^j L(\cdot)$ *are the partial gradients of* $L(\cdot)$ *with respect to the parameters in* i*-th row and* j*-th column, respectively. We consider conditions (ii) and (iv) since Lemma 3.13 handles the condition for zero parameters. By using the covariate duplication method (Obozinski et al., 2011), if* $\boldsymbol{V}_{(i)} = \boldsymbol{0}$*,* $\nabla_i L(\boldsymbol{W})$ *in condition (ii) is computed as follows:*

$$\nabla_i L(\boldsymbol{W}) = \boldsymbol{X}^{(i)\mathrm{T}}\{\boldsymbol{X} - (\boldsymbol{X}\boldsymbol{W} - \boldsymbol{X}^{(i)}\boldsymbol{V}_{(i)})\boldsymbol{X}\}\boldsymbol{X}^{\mathrm{T}}. \tag{3.19}$$

Therefore, we obtain the condition $R_i \leq \lambda_r$ *for* $\boldsymbol{V}_{(i)} = \boldsymbol{0}$ *in Lemma 3.13 from* $\|\nabla_i L(\boldsymbol{W})\|_2 = R_i$ *and condition (ii). Similarly, if* $\boldsymbol{H}^{(j)} = \boldsymbol{0}$, $\nabla^j L(\boldsymbol{W})$ *in condition (iv) is computed as follows:*

$$\nabla^j L(\boldsymbol{W}) = \boldsymbol{X}^{\mathrm{T}}\{\boldsymbol{X} - \boldsymbol{X}(\boldsymbol{W}\boldsymbol{X} - \boldsymbol{H}^{(j)}\boldsymbol{X}_{(j)})\}\boldsymbol{X}_{(j)}^{\mathrm{T}}. \tag{3.20}$$

From $\|\nabla^j L(\boldsymbol{W})\|_2 = C_j$ *and condition (iv), we obtain the condition* $C_j \leq \lambda_c$ *in Lemma 3.13.* □

Proof of Theorem 3.1

Proof *The precomputations of* $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ *and* $\|\boldsymbol{G}_{(i)}\|_2$ *for all the rows require* $\mathcal{O}(p^2 n)$ *and* $\mathcal{O}(p^2)$ *times, respectively. Since the lower bound* $\underline{K}_i$ *is computed at* $\mathcal{O}(1)$ *time from Lemma 3.11, the construction of the set* $\mathbb{M}$ *requires* $\mathcal{O}(p)$ *time as shown in Lemma 3.12. According to the sequential rule, the total cost of the construction is* $\mathcal{O}(pQ)$ *time. Since computation of all the rows of* $\tilde{K}_i$ *requires* $\mathcal{O}(p^2 n)$ *time, the total cost is* $\mathcal{O}(p^2 n t_u)$ *time for all the outer loops of the coordinate descent with the upper bounds. From Lemma 3.6, the total computation cost of the upper bounds* $\overline{K}_i$ *is* $\mathcal{O}(p^2 t_u)$ *because* $\mathcal{O}(p^2)$ *time is required to compute the upper bounds of all the rows. To update the parameter vectors, we need* $\mathcal{O}(pnt_m)$ *time for the coordinate descent on the set* $\mathbb{M}$*. For the coordinate descent with the upper bounds,* $\mathcal{O}(p^2 n t_u S)$ *time is required for the updates. This is because the total number of inner loops is* pt_u*, and the updates are performed only when they are unskipped by the upper bound. Thus, Algorithm 3.2 needs* $\mathcal{O}(p\{n(t_m + pt_u S) + Q\})$ *time.* □

Proof of Theorem 3.2

Proof *Since we assume that the coordinate descent converges to a global optimal solution, our algorithm converges to a global optimal solution at line 13 of Algorithm 3.2. In addition, from Lemma 3.4, Algorithm 3.2 safely skips unnecessary updates of the coordinate descent (lines 20–21). Since we assume that the coordinate descent converges to a global optimal solution in the theorem, Algorithm 3.2 converges to the same objective values as those of the original algorithm at line 25 if their regularization constants are the same.* □

Part II

Fast Algorithms for Deep Models

Chapter 4

Fast Gradient Descent with Adaptive Learning Rate for Deep Neural Networks

4.1 Introduction

Adaptive learning rate algorithms are widely used for efficient training of deep neural networks. RMSProp (Tieleman and Hinton, 2012) and its follow-on methods such as AdaDelta (Zeiler, 2012) and Adam (Kingma and Ba, 2014) are being used in many deep neural networks such as Convolutional Neural Networks (CNNs) (LeCun et al., 1998) since they can be easily implemented with high memory efficiency.

However, the gradients used in RMSProp include noise caused by stochastic optimization techniques such as mini-batch setting. With batch setting, since the model inputs are fixed in each iteration, only parameter updates change the gradients. On the other hand, with mini-batch setting, since the inputs are not fixed in each iteration, gradients can also be changed by randomly selecting the inputs in each iteration. This change in the mini-batch setting can be seen as noise. Although RMSProp uses noisy gradients, it does not consider the noise when it adjusts the learning rate. This indicates that the efficiency of RMSProp can be improved by effectively handling noise in the gradients.

This chapter proposes a novel adaptive learning rate algorithm called *SDProp*.

The key idea is to utilize covariance matrix based preconditioning to effectively handle noise in the gradients. The covariance matrix is derived by assuming a distribution for the noise in the observed gradients. Since the distribution effectively captures the noise, SDProp can effectively capture the changes in gradients caused by random input selection in each iteration. In experiments, we compare SDProp with RMSProp. SDProp needs 50 % fewer training iterations than RMSProp to reach the final training loss for CNN in CIFAR-10, CIFAR-100 and MNIST datasets. In addition, SDProp outperforms Adam, a state-of-the-art algorithm based on RMSProp, in several datasets. Our approach is also more effective than RMSProp for training Recurrent Neural Network (RNN) (Elman, 1990) and very deep fully-connected neural networks.

4.2 Preliminary

We briefly review the background of this chapter. First, we describe SGD, which is a basic algorithm in stochastic optimization for training deep neural networks. Next, we review RMSProp that can effectively train deep neural networks on the basis of SGD and the adaptive learning rate.

4.2.1 Stochastic Gradient Descent

Many learning algorithms aim at minimizing loss function $f(\theta)$ with respect to parameter vector θ. SGD is a popular algorithm in the mini-batch setting. To minimize $f(\theta)$, SGD iteratively updates θ with a mini-batch of samples as follows:

$$\theta_i^{(t)} = \theta_i^{(t-1)} - \alpha \nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)}) \tag{4.1}$$

where α is the learning rate, $\theta_i^{(t)}$ is the i-th element of the parameter vector at time t, $x^{(t-1)}$ is the sample or mini-batch at time $t-1$, and $\nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)})$ is the gradient with respect to the i-th parameter given by $\theta^{(t-1)}$ and $x^{(t-1)}$. SGD applies Equation (4.1) to each sample or mini-batch while Gradient Descent (GD) applies Equation (4.1) to all data in the batch setting. Although $\nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)})$ includes noise due to the random selection of mini-batch $x^{(t-1)}$, SGD uses it in the training phase. Since SGD only uses a part of the data for computing $\nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)})$,

each iteration has reduced computation cost while memory efficiency is high.

4.2.2 RMSProp

RMSProp is a popular algorithm based on SGD for training neural networks. AdaDelta and Adam are follow-up methods of RMSProp. RMSProp rapidly reduces loss function $f(\theta)$ by adapting the learning rate of SGD. The updating rule of RMSProp is as follows:

$$v_i^{(t)} = \beta v_i^{(t-1)} + (1-\beta)\left(\nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)})\right)^2 \tag{4.2}$$

$$\theta_i^{(t)} = \theta_i^{(t-1)} - \frac{\alpha}{\sqrt{v_i^{(t)}+\epsilon}} \nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)}) \tag{4.3}$$

where $v_i^{(t)}$ is the moving average of squared gradients $\left(\nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)})\right)^2$, β is the decay rate for computing $v_i^{(t)}$, and ϵ is a small positive value for the stable computation. Intuitively, RMSProp divides the learning rate, α, by magnitude $\sqrt{v_i^{(t)}}$ of the past gradients. Therefore, if the i-th parameter has large gradients in terms of the magnitude in the past, RMSProp yields a small learning rate because $\sqrt{v_i^{(t)}}$ in Equation (4.3) is large. Empirically, this idea efficiently reduces the value of the loss function for deep neural networks. Follow-up methods such as AdaDelta and Adam are based on this idea.

4.3 Proposed Method

We first introduce our idea of the covariance matrix based preconditioning. Then, we introduce our algorithm on the basis of this idea. All the proofs can be found in Section 4.7.

4.3.1 Idea

RMSProp utilizes the moving average of squared gradients to adjust the learning rate as described in the preliminary section. However, in stochastic optimization approaches, the gradients include noise because input $x^{(t-1)}$ is randomly selected in each iteration. Since RMSProp does not consider the noise in the gradients, it may be difficult to effectively adjust the learning rate. In order to handle the noise,

we propose SGD with preconditioning on the basis of the covariance matrix of the gradients.

In the covariance matrix based preconditioning, we assume that the gradients follow a Gaussian distribution. By following (Sra et al., 2012), we assume the following Gaussian distribution of gradient $\hat{g}_t = \nabla_\theta f(\theta^{(t)}; x^{(t)}) \in \mathbb{R}^d$:

$$\hat{g}_t | \bar{g}_t \sim N(\bar{g}_t, C_t) \tag{4.4}$$

where $\bar{g}_t \in \mathbb{R}^d$ is the true gradient without the noise while $\hat{g}_t \in \mathbb{R}^d$ includes the noise. $N(\bar{g}_t, C_t)$ is a Gaussian distribution with mean $\bar{g}_t$ and covariance matrix C_t; C_t is the covariance matrix of $\hat{g}_t$ whose size is $d \times d$. The diagonal elements in C_t represent the magnitude of oscillation of the gradients $\hat{g}_t$ that include the noise. Specifically, let $C_t[i, j]$ be the i-th row and the j-th column element in C_t, $C_t[i, j]$ represents the covariance of the i-th and the j-th gradient. Therefore, if the i-th gradient strongly correlates with the j-th gradient, $C_t[i, j]$ has large absolute value. On the other hand, $C_t[i, i]$ represents the variance of the i-th gradient. Therefore, $C_t[i, i]$ has large value if the gradient strongly oscillates in the i-th dimension.

Intuitively, large oscillations in i-th dimension incur high variance of updating directions and inefficient progress in plain SGD. However, it is difficult to reduce the oscillation since it can be a result of the noise induced by the mini-batch setting. How can we reduce the oscillation by using C_t ? This is the motivation behind our approach; we assume that plain SGD efficiently progresses if we can control the oscillation by utilizing C_t. In this chapter, we propose the preconditioning of C_t to control the oscillation. Namely, we reduce the condition number of C_t by utilizing preconditioning on the basis of the covariance matrix of the gradients. In particular, our preconditioning transforms C_t into an identity matrix. We describe our approach in the next section.

4.3.2 Covariance Matrix Based Preconditioning

The previous section suggests that large values in the diagonal of C_t prevent the efficient progress of SGD. Therefore, if we could control the values in the diagonal of C_t, we improve the efficiency of SGD. Our covariance matrix based preconditioning transforms C_t into $\rho^2 I$ where I is an identity matrix whose size is $d \times d$ and ρ is a

hyper-parameter that has a positive value. Since the element in the diagonal of C_t represents the variance of the gradients, we can hold the variance to constant value ρ^2. If the variance is larger than ρ^2, its value is reduced to ρ^2.

We first describe the approach used to transform C_t into I instead of $\rho^2 I$. This is because once C_t is transformed into I, it is easy to transform I into $\rho^2 I$ as we describe later. To execute the preconditioning of C_t, we use the transformation $g_p = D^{-1}\hat{g}_t$ where $D \in \mathbb{R}^{d \times d}$ is called as the preconditioning matrix. In this transformation, g_p is a transformed gradient and $\hat{g}_t$ is a gradient as defined in Equation (4.4). Since the transformation is an affine transformation of $\hat{g}_t$ generated from the Gaussian distribution in Equation (4.4), we have following distribution of g_p:

$$g_p = D^{-1}\hat{g}_t | \bar{g}_t \sim N(D^{-1}\bar{g}_t, D^{-1}C_t(D^{-1})^{\mathrm{T}}). \tag{4.5}$$

In Equation (4.5), we use the following major rule to transform Equation (4.4) into (4.5): if $X \sim N(m, \Sigma)$ and $Y = AX$, then $Y \sim N(Am, A\Sigma A^T)$; $N(m, \Sigma)$ is a Gaussian distribution that has mean m and covariance matrix Σ, A is a matrix for affine transformation and Y is a transformed variable. By setting $D = C_t^{1/2}$ in Equation (4.5), we have the following property :

Theorem 4.1 *If we transform gradient $\hat{g}_t$ to yield $g_p = C_t^{-1/2}\hat{g}_t$, we have the following Gaussian distribution:*

$$g_p | \bar{g}_t \sim N(C_t^{-1/2}\bar{g}_t, I) \tag{4.6}$$

where I is an identity matrix whose size is $d \times d$.

The above theorem indicates that the transformation of $g_p = C_t^{-1/2}\hat{g}$ results in the Gaussian distribution of g_p whose covariance matrix is identity matrix I. In other words, we can control the covariance matrix to be I by using g_p instead of $\hat{g}$.

Our preconditioning transforms the value of variance for gradients into 1 by using g_p. However, g_p may have an extremely large value if the variance is 1. Thus, we introduce hyper-parameter ρ to generalize our preconditioning. Specifically, by using the transformation of $g_p = \rho C_t^{-1/2}\hat{g}$ instead of $g_p = C_t^{-1/2}\hat{g}$, we have the following distribution:

$$\rho C_t^{-1/2}\hat{g}_t | \bar{g}_t \sim N(\rho C_t^{-1/2}\bar{g}_t, \rho^2 I). \tag{4.7}$$

The above equation denotes that ρ controls the value of the covariance matrix while the previous transformation only gives an identity matrix as shown in Equation (4.6). We show that ρ has the same role as learning rate α when we derive SDProp in the next section.

Since we compute the gradients at each time t in SGD, we have to incrementally compute covariance matrix C_t although Theorem 4.1 is based on the property that C_t is a positive semi-definite matrix. In order to incrementally compute C_t as a positive semi-definite matrix, we use the online updating rule of (Sra et al., 2012) as follows:

$$C_t = \gamma C_{t-1} + \gamma(1-\gamma)(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}} \tag{4.8}$$

$$\mu_t = \gamma \mu_{t-1} + (1-\gamma)\hat{g}_t \tag{4.9}$$

where μ_t is the moving average of $\hat{g}_t$ and γ is the hyper-parameter of the decay rate for the moving average that has $\gamma \in [0, 1)$. C_t and μ_t are initialized as $\mu_1 = \hat{g}_1$ and $C_1 = 0$. The above updating rule gives the following property:

Theorem 4.2 *If we compute covariance matrix C_t by using Equations (4.8) and (4.9), C_t is positive semi-definite.*

Thus, if we compute C_t by using Equations (4.8) and (4.9), we can execute the preconditioning specified by Theorem 4.1.

4.3.3 Algorithm

Since deep neural networks have a large number of parameters, the idea described in the previous section incurs large memory consumption of $O(d^2)$ where d is the number of parameters. In addition, it costs $O(d^3)$ time to compute $D = C_t^{1/2}$ by using eigenvalue decomposition (Halko et al., 2011). To avoid these problems, we employ diagonal preconditioning matrix $D = \mathrm{diag}(C_t)^{1/2}$. Since this approach only needs the diagonal terms, the memory and computation costs are $O(d)$. Although this approach ignores the correlation of gradients, it is sufficient to control the oscillation in each dimension. This is because the diagonal of C_t represents the variance of the oscillation as described in the previous section. By picking the diagonal of Equation (4.7), the

updating rule is:

$$\theta^{(t)} = \theta^{(t-1)} - \rho \cdot \mathrm{diag}\,(C_t)^{-1/2}\, \nabla_\theta f(\theta^{(t-1)}; x^{(t-1)}). \tag{4.10}$$

We rewrite this updating rule (all steps) as follows:

$$\mu_i^{(t)} = \gamma \mu_i^{(t-1)} + (1-\gamma) \nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)}) \tag{4.11}$$

$$c_i^{(t)} = \gamma c_i^{(t-1)} + \gamma(1-\gamma)(\nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)}) - \mu_i^{(t-1)})^2 \tag{4.12}$$

$$\theta_i^{(t)} = \theta_i^{(t-1)} - \frac{\rho}{\sqrt{c_i^{(t)}} + \epsilon} \nabla_{\theta_i} f(\theta^{(t-1)}; x^{(t-1)}) \tag{4.13}$$

where $\mu_i^{(t)}$ is the moving average of gradients for the i-th parameter at time t and γ is the hyper-parameter of the decay rate for the moving average that has $\gamma \in [0, 1)$. $c_i^{(t)}$ is the exponentially moving variance of gradients for the i-th parameter at time t. We use γ in Equation (4.12) as the decay rate of the exponentially moving variance. $\mu_i^{(t)}$ and $c_i^{(t)}$ are initialized as 0. For stable computation, ϵ is set at a small positive value. Equation (4.13) corresponds to Equation (4.10). We call the algorithm *SDProp* because Equation (4.13) includes *Standard Deviation* $\sqrt{c_i^{(t)}}$. Although $c_i^{(t)}$ includes the bias imposed by initialization, we can remove the bias in the same way as (Kingma and Ba, 2014).

Notice that ρ takes the same role as learning rate α in Equation (4.3) of RMSProp. Therefore, Equation (4.13) divides the learning rate by the square root of *centered* variance $c_i^{(t)}$ while Equation (4.3) of RMSProp divides the learning rate by the square root of *uncentered* variance $v_i^{(t)}$. In other words, RMSProp and its follow-up methods such as Adam adapt the learning rate by the magnitude of gradients while we adapt it by the variance of gradients. Although RMSProp and SDProp have similar updating rules, they have totally different goals as described in the previous sections.

4.4 Experiments

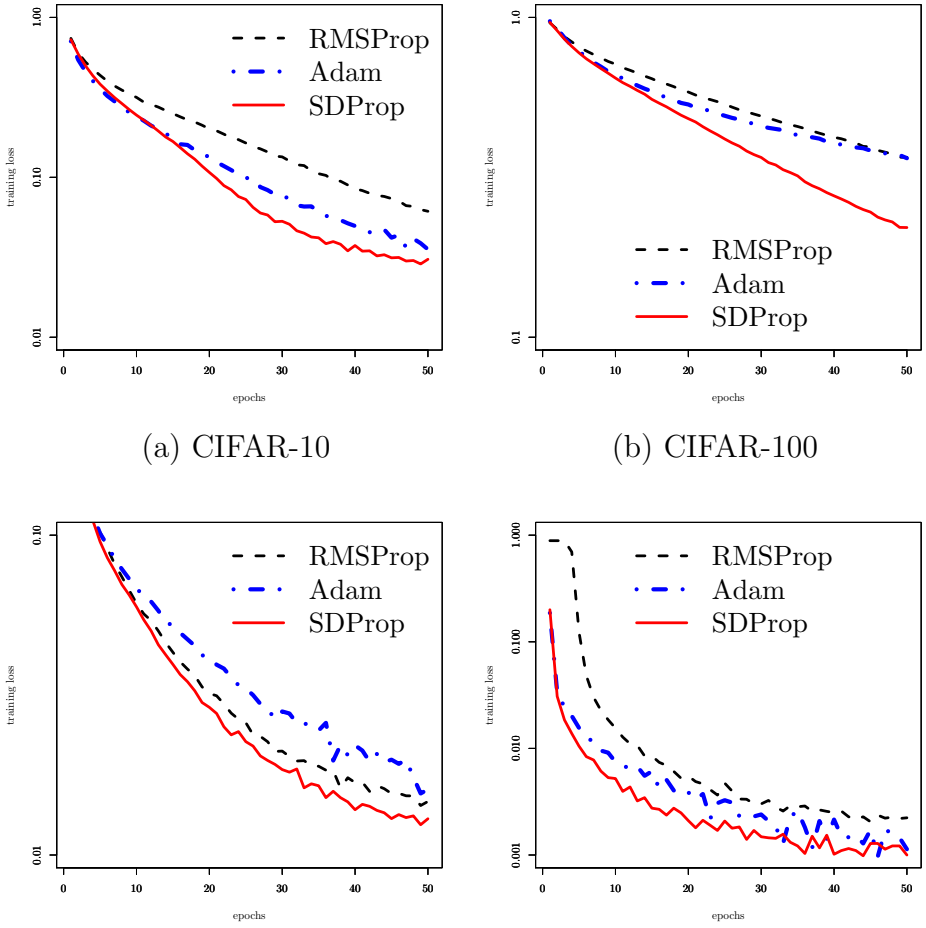
We performed experiments to compare SDProp to RMSProp and Adam, a state-of-the-art algorithm based on RMSProp. Kingma and Ba (2014) show that Adam is a more efficient and effective approach than RMSProp or AdaDelta by integrating momentum into RMSProp. First, we show the efficiency and effectiveness of our

approach by using CNN. Second, since SDProp effectively handles the oscillation described in the previous section, we evaluate SDProp by using small mini-batches which suffer noise in the gradients. Third, we show the efficiency and effectiveness of SDProp for RNN. Fourth, we demonstrate the effectiveness of SDProp for 20 layered fully-connected neural network.

4.4.1 Efficiency and Effectiveness for CNN

We investigate the efficiency and effectiveness of SDProp. We used 4 datasets to assess the classification of images; CIFAR-10, CIFAR-100 (Krizhevsky and Hinton, 2009), SVHN (Sermanet et al., 2012) and MNIST. The experiments were conducted on a 7-layered CNN with ReLU activation function. The loss function was negative log likelihood. We compared SDProp to RMSProp and Adam. In SDProp, we tried various combinations of hyper-parameters by using $\gamma \in \{0.9, 0.99\}$ and $\rho \in \{0.1, 0.01, 0.001\}$. In RMSProp, we tried combinations of hyper-parameters by using $\beta \in \{0.9, 0.99\}$ and $\alpha \in \{0.1, 0.01, 0.001\}$. As a result, SDProp achieves the lowest loss in the settings of $\gamma = 0.99$, $\rho = 0.001$. RMSProp has the lowest loss when $\beta = 0.99$ and $\alpha = 0.001$. Adam achieves the lowest loss when $\beta_1 = 0.9$, $\beta_1 = 0.999$ and $\alpha = 0.001$. The mini-batch size was 128. The number of epochs was 50. We use the training loss to evaluate the algorithms because they optimize the training criterion.

Figure 4.1 shows the training losses of each dataset. In CIFAR-10, CIFAR-100 and SVHN, SDProp yielded lower losses than RMSProp and Adam in early epochs. In MNIST, although the training loss of SDProp and Adam nearly reached 0.0, SDProp reduces the loss faster than Adam. SDProp needs 50 % fewer training iterations than RMSProp to reach its final training loss in CIFAR-10, CIFAR-100 and MNIST. Since SDProp effectively captures the noise, it effectively reduces the loss even if the gradients are noisy. In the next experiment, we investigate the performance of SDProp in terms of its effectiveness against noise by using noisy gradients.



(a) CIFAR-10 (b) CIFAR-100

(c) SVHN (d) MNIST

Figure 4.1: Training losses for CNN. We show the results for (a) CIFAR-10, (b) CIFAR-100, (c) SVHN and (d) MNIST.

Table 4.1: Training accuracy percentage for CIFAR-10 in CNN for different mini-batch sizes. We tuned the hyper-parameters; the 1st row presents mini-batch size.

	16	*32*	*64*	*128*
RMSProp	81.42	93.10	94.98	95.07
Adam	83.24	93.57	95.48	97.12
SDProp	**90.17**	**94.87**	**96.54**	**97.31**

4.4.2 Sensitivity of Mini-batch Size

The previous experimental results show that SDProp is more efficient and effective than existing methods because it well handles the noise in our idea and in practice. In other words, SDProp is expected to effectively train the model even if we use small mini-batch sizes that incur noisy gradients (Dekel et al., 2012). Therefore, we investigated the sensitivity of SDProp and existing methods to mini-batch size. While the main purpose of this experiment is to reveal the one performance attribute of SDProp, the result suggests that SDProp can be used on devices with scant memory that must use small mini-batches.

We compared SDProp to RMSProp and Adam using mini-batch sizes of 16, 32, 64 and 128. We used the CIFAR-10 dataset for the 10-class image classification task. We used CNN as per the previous section. The hyper-parameters are also the same as the previous section; they are tuned by grid search. The number of epochs was 50.

Table 4.1 shows the final training accuracies. SDProp outperforms RMSProp and Adam in all mini-batch size values examined. Specifically, although small mini-batch size of 16 incurs very noisy gradients, SDProp obviously achieves effective training unlike RMSProp and Adam. In addition, Table 4.1 shows that the superiority of our approach over RMSProp and Adam increases as mini-batch size falls. For example, if the mini-batch size is 16, our approach has 8.75 percent higher accuracy than RMSProp and 2.24 percent more accurate if the mini-batch size is 128. This indicates that our covariance matrix based preconditioning effectively handles the noise of gradients.

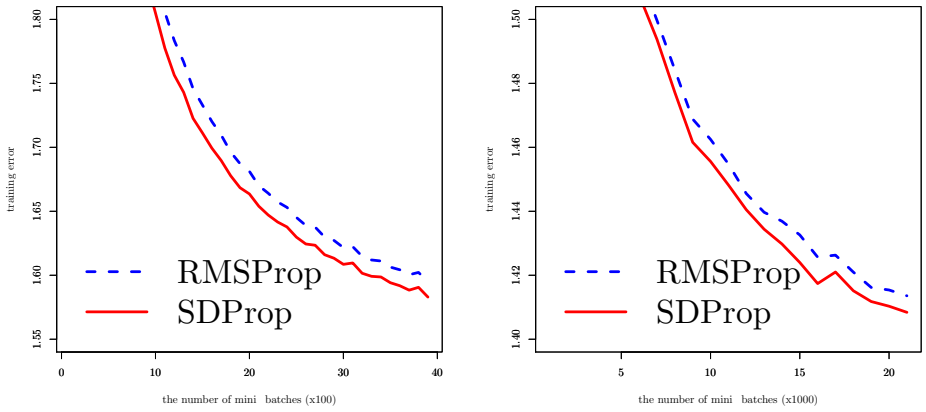


Figure 4.2: Cross entropies in training RNN for shakespeare dataset (left) and source code of linux kernel (right).

4.4.3 Efficiency and Effectiveness for RNN

We evaluated the efficiency and effectiveness of SDProp for the Recurrent Neural Network (RNN). In this experiment, we predicted the next character by using previous characters via character-level RNN. We used the subset of shakespeare dataset and the source code of the linux kernel as the dataset (Karpathy et al., 2015). The size of the internal state was 128. The pre-processing of the dataset followed that of (Karpathy et al., 2015). The mini-batch size was 128. In SDProp, we tried grid search with $\rho \in \{0.1, 0.01, 0.001\}$ and $\gamma \in \{0.9, 0.99\}$. As a result, SDProp used the settings of $\rho = 0.01$ and $\gamma = 0.99$. In RMSProp, we tried grid search with $\alpha \in \{0.1, 0.01, 0.001\}$ and $\beta \in \{0.9, 0.99\}$. Finally, we used the settings of $\alpha = 0.01$ and $\beta = 0.99$ for RMSProp. The training criterion was cross entropy. We used gradient clipping and learning rate decay. Gradient clipping is a popular approach for scaling down the gradients by manually setting a threshold; it prevents gradients from exploding in RNN training (Pascanu et al., 2013). We set the threshold to 5.0. We decayed the learning rate α every tenth epoch by the factor of 0.97 for RMSProp following (Karpathy et al., 2015). In SDProp, ρ was also decayed the same as α of RMSProp.

Figure 4.2 shows the results of the shakespeare dataset and the source code of the linux kernel. SDProp reduces the training loss faster than RMSProp. Since SDProp effectively handles the noise induced by the mini-batch setting, it can efficiently train

Table 4.2: Average, Best and Worst training accuracy percentage of 20 layered fully-connected networks.

			Accuracy		
Method	α	β	Ave.	Best	Worst
RMSProp	0.001	0.9	92.86	97.95	84.79
		0.99	98.81	99.11	98.34
	ρ	γ	Ave.	Best	Worst
SDProp	0.001	0.9	93.77	97.9	87.57
		0.99	**99.20**	**99.42**	**99.09**

models other than CNN, such as RNN.

4.4.4 20 Layered Fully-connected Neural Network

In this section, we performed experiments to evaluate the effectiveness of SDProp for training deep fully-connected neural networks. Dauphin et al. (2014) suggests that the number of saddle points exponentially increases with the dimensions of the parameters. Since deep fully-connected networks typically have parameters with higher dimension than other models such as CNN, this optimization problem has many saddle points. This problem is challenging because SGD slowly progresses around saddle points (Dauphin et al., 2014).

We used a very deep fully-connected network with 20 hidden layers, 50 hidden units and ReLU activation functions. We used the MNIST dataset for the 10-class image classification task. This setting is the same as (Neelakantan et al., 2015) used in evaluating the effectiveness of SGD with high dimensional parameters. The training criterion was negative log likelihood. The mini-batch size was 128. We initialized parameters from a Gaussian with mean 0 and standard deviation 0.01 following Neelakantan et al. (2015). We compared SDProp to RMSProp. In SDProp, we tried the combinations of hyper-parameters by using $\gamma \in \{0.9, 0.99\}$ and $\rho \in \{0.1, 0.01, 0.001\}$. In RMSProp, we tried the combinations of hyper-parameters by using $\beta \in \{0.9, 0.99\}$ and $\alpha \in \{0.1, 0.01, 0.001\}$. The number of epochs was 50. We tried 10 runs for each of the above settings.

Table 4.2 lists the results for the best setting of α and ρ. It shows averages, best, worst of training accuracies for each setting. The result shows that SDProp achieves higher accuracy than RMSProp for the best setting. In addition, the difference

between best and worst accuracy of SDProp is smaller than RMSProp. Since SDProp effectively handles the randomness of noise, it can reduce result uncertainty. The results show that SDProp effectively trains models that have very high dimensional parameters.

4.5 Discussion

This chapter proposed a novel preconditioning on the basis of the covariance matrix of gradients. This section explains the relationship between TONGA (Roux et al., 2008), which is closely related to this study. TONGA is another major covariance matrix based preconditioning that assumes a prior distribution for the gradients. It computes the posterior distribution of the gradients and updates the parameters on the basis of the posterior distribution. As a result, the covariance matrix is used as the preconditioning matrix in TONGA. Interestingly, Roux et al. (2008) revealed that the updating rule of TONGA is almost identical to that of natural gradient (Amari, 1998). On the other hand, our method uses the square root of the covariance matrix as the preconditioning matrix. Although this difference seems to be trivial, TONGA and our method are proposed for different purposes. Namely, our method transforms the covariance matrix into ρI as described in Section 4.3.2 while TONGA does not. In addition, the updating rule of our method is different from that of natural gradient unlike TONGA as it utilizes the square root of the covariance matrix. Although our method can be regarded as whitening of gradients, more research is still needed to conclude why whitening of gradients improves the efficiency of training DNNs.

4.6 Summary

We proposed SDProp for the effective and efficient training of deep neural networks. Our approach utilizes the idea of using covariance matrix based preconditioning to effectively handle the noise present in the gradients. Our experiments showed that, for various datasets and models, SDProp is more efficient and effective than existing methods. In addition, SDProp achieved high accuracy even if the gradients were noisy.

4.7 Proofs

Proof of Theorem 4.1

Proof *By using eigen decomposition, we can represent* C_t *as* $C_t = U\Sigma U^{\mathrm{T}}$ *where* U *is an orthogonal matrix of* $d \times d$ *and* Σ *is a diagonal matrix of* $(\lambda_1, \lambda_2, ..., \lambda_d)$. *Since* C_t *is assumed to be a positive semi-definite matrix, all eigen values are equal to or higher than 0. Thus,* $C_t^{1/2}$ *can be computed as* $C_t^{1/2}{=}U\Sigma^{1/2}U^{\mathrm{T}}$. *By setting the covariance term of Equation (4.5) to* $D^{-1}{=}(C_t^{1/2})^{-1}{=}U\Sigma^{-1/2}U^{\mathrm{T}}$, *the Gaussian distribution of* g_p *is represented as follows:*

$$
\begin{aligned}
&g_p = C_t^{-1/2}\hat{g}_t|\bar{g}_t \sim N(C_t^{-1/2}\bar{g}_t, C_t^{-1/2}C_t(C_t^{-1/2})^{\mathrm{T}}) \\
&= N(C_t^{-1/2}\bar{g}_t, U\Sigma^{-1/2}U^{\mathrm{T}}U\Sigma U^{\mathrm{T}}(U\Sigma^{-1/2}U^{\mathrm{T}})^{\mathrm{T}}) \\
&= N(C_t^{-1/2}\bar{g}_t, I).
\end{aligned}
$$

In the above formulations, since U *is an orthogonal matrix, we use* $UU^{\mathrm{T}}{=}\ I$ *and* $(U\Sigma^{-1/2}U^{\mathrm{T}})^{\mathrm{T}}{=}U\Sigma^{-1/2}U^{\mathrm{T}}$. *As a result, we have the distribution of Equation (4.6).*

□

Proof of Theorem 4.2

Proof *In order to prove Theorem 4.2, we first prove that* $(\hat{g}_t{-}\mu_{t-1})(\hat{g}_t{-}\mu_{t-1})^{\mathrm{T}}$ *in Equation (4.8) is a positive semi-definite matrix. By setting* $y = x^{\mathrm{T}}(\hat{g}_t{-}\mu_{t-1})$, *we have:*

$$x^{\mathrm{T}}(\hat{g}_t{-}\mu_{t-1})(\hat{g}_t{-}\mu_{t-1})^{\mathrm{T}}x = yy^{\mathrm{T}} \geq 0.$$

By following the definition of positive semi-definite matrixes, if we have matrix A *of* $d \times d$ *such that* $x^{\mathrm{T}}Ax \geq 0$ *holds for every non-zero column vector* x *of* d *real numbers,* A *is a positive semi-definite matrix. Since the above inequation shows that* $x^{\mathrm{T}}(\hat{g}_t{-}\mu_{t-1})(\hat{g}_t{-}\mu_{t-1})^{\mathrm{T}}x \geq 0$ *holds, it is clear that* $(\hat{g}_t{-}\mu_{t-1})(\hat{g}_t{-}\mu_{t-1})^{\mathrm{T}}$ *is a positive semi-definite matrix even if* μ_{t-1} *in Equation (4.9) has any real value.*

Then, we prove that C_t *in Equation (4.8) is a positive semi-definite matrix by mathematical induction.*

Initial step: If t=1, *the initialization yields* $C_1 = 0$. *Since* C_2 *is computed as* $C_2 = \gamma(1-\gamma)(\hat{g}_2 - \mu_1)(\hat{g}_2 - \mu_1)^{\mathrm{T}}$ *by using Equation (4.8) and (4.9),* C_2 *is a positive semi-definite matrix. This is because* $(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}$ *is a positive semi-definite matrix as proved above.*

Inductive step: We assume that C_{t-1} *is a positive semi-definite matrix. Since* C_t *is computed as* $C_t = \gamma C_{t-1} + \gamma(1-\gamma)(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}$ *by using Equations (4.8) and (4.9),* $x^{\mathrm{T}} C_t x$ *is represented as follows:*

$$\begin{aligned} x^{\mathrm{T}} C_t x &= x^{\mathrm{T}}(\gamma C_{t-1} + \gamma(1-\gamma)(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}})x \\ &= \gamma x^{\mathrm{T}} C_{t-1} x + \gamma(1-\gamma) x^{\mathrm{T}}(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}} x. \end{aligned}$$

In the above equation, $x^{\mathrm{T}} C_{t-1} x \geq 0$ *and* $x^{\mathrm{T}}(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}} x \geq 0$ *because* C_{t-1} *and* $(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}$ *are positive semi-definite matrices. Therefore,* C_t *is a positive semi-definite matrix because* $x^{\mathrm{T}} C_t x \geq 0$ *holds in the above equation. This completes the inductive step.* □

Chapter 5

Inference Acceleration with Layer-Wise Model Compression for Residual Networks

5.1 Introduction

Convolutional Neural Networks (CNNs) (LeCun et al., 1998) are important tools in the machine learning community because they have a wide field of applications. Although modern CNNs need a lot of time for the training phase, it can be shortened by using many computation resources. In fact, Akiba et al. (2017) shows that large CNNs can be trained within 15 minutes on ImageNet datasets by using 1024 GPUs. On the other hand, after the training phase, the inference phase is used to perform prediction in service deployment. Since the era of IoT has arrived, it is increasingly important to perform inference on devices with limited resources such as image classification on embedded systems (Wu et al., 2016), character recognition on portable devices (Xiao et al., 2017) and speech recognition on mobile devices (Schuster, 2010).

While we need to perform inference with limited resources, the number of layers in CNNs has been increasing every year in order to raise accuracy. In 1998, LeNet-5 used 5 layers to classify handwriting digits (LeCun et al., 1998). In 2012, AlexNet won an ILSVRC image classification competition with 8 layers (Krizhevsky et al., 2012). In the competition of 2014, VGG Net and GoogleNet stacked 19 and 22 layers,

respectively (Simonyan and Zisserman, 2014; Szegedy et al., 2015). Residual Network (ResNet) used 152 layers and won the competition in 2015 (He et al., 2016b). The paper of ResNet has many citations, more than 5,000 just within the last two years, and ResNet is being used as a standard CNN-based model.

However, due to its sheer number of layers, ResNet incurs considerable computation overheads such as processing time and memory usage. Although we can efficiently train CNNs by using GPUs, it is still difficult to perform inference efficiently with limited resources such as embedded systems and mobile devices. Several approaches reduce the number of layers in performing the inference phase (Larsson et al., 2017; Ba and Caruana, 2014; Zagoruyko and Komodakis, 2016; Veit and Belongie, 2018; Wu et al., 2018; Huang and Wang, 2018). Unfortunately, they incur additional memory requirements or degrade the accuracy on real-world datasets such as ImageNet. Therefore, we need other strategies that reduce the number of layers without increasing memory consumption while keeping accuracy high.

To achieve this goal, we propose Network Implosion (NI); it erases multiple layers from ResNet without increasing the computation costs in the inference phase. Our proposal introduces a priority term that indicates the importance of each layer. We can select and erase unimportant layers according to the priority after training the network. In addition, our method can avoid any critical drop in accuracy by retraining the network with large learning rate after erasure. Our experiments show that NI reduces the number of layers in ResNet with no additional computation costs in the inference; for classification tasks on CIFAR-10/100 and ImageNet, the layer reductions are 57.14%∼76.00%.

5.2 Related work

Dynamic Layer Pruning. Veit et al. (2016) found that ResNet does not suffer a significant loss of accuracy if a few layers are erased. In their experiments, when layers were erased from networks, ResNet suffered only a slight drop in accuracy while the accuracy of the VGG architecture (Simonyan and Zisserman, 2014) dropped significantly in the inference phase. Inspired by the results gained, some recent papers erase layers from ResNet in order to raise processing speed. Wu et al. (2018); Veit and Belongie (2018) dynamically erase layers that are not needed during the inference

phase. These "Dynamic Layer Pruning" can easily keep the accuracy, however, they can not reduce the memory consumption.

Static Layer Pruning. "Static Layer Pruning" completely erases layers while Dynamic Layer Pruning only selects layers to be removed during the inference. Thus Static Layer Pruning is preferable in terms of the computation cost for the inference because it reduces the memory consumption while Dynamic Layer Pruning cannot. Wen et al. (2016); Huang and Wang (2018) utilize sparse regularizations (Fujiwara et al., 2016a;b) that erase layers from ResNet. Yu et al. (2018) ignores layers that have subthreshold activations. However, since these methods need to tune the continuous hyper parameters of the regularization or the threshold, it is difficult to obtain the desired number of layers.

Teacher-Student Training. The teacher-student training regime is a well-known algorithm that trains shallow student networks by using deep trained teacher networks (Ba and Caruana, 2014; Hinton et al., 2015). In teacher-student training, the shallow student networks can be effectively trained because the deep trained teacher network gives the probability distribution over the classes to the student networks in order to boost their training. Since we can freely design the student network, we can reduce the number of layers without additional computation costs in the inference. In addition, Hinton et al. (2015) reports that teacher-student training improves the accuracy on several datasets and tasks.

Other several papers also try to reduce the number of layers in deep neural networks. FractalNet (Larsson et al., 2017) can reduce the number of layers for the inference phase without increasing the parameters; unfortunately, it degrades the accuracy when a 20-layer model was used instead of a 40-layer model. Zagoruyko and Komodakis (2016) showed how to achieve high accuracy by increasing the number of parameters in each layer even if the network is shallow. However, their 50-layer model has more parameters than the usual 200-layer model.

5.3 Preliminary

This section introduces ResNet, which is now widely used as a standard CNN-based model. ResNets have blocks called Residual Units (He et al., 2016b). Since each Residual Unit has multiple convolutional layers, ResNets can have deep architectures

by stacking Residual Units. The l-th Residual Unit introduced in (He et al., 2016a) is defined as follows:

$$\mathbf{x}_{l+1} = \mathbf{x}_l + F(\mathbf{x}_l), \tag{5.1}$$

where $\mathbf{x}_l$ is the input to the l-th Residual Unit. $F(\cdot)$ is a module that consists of convolutional layers, batch normalizations (Ioffe and Szegedy, 2015) and Rectified Linear Units (ReLUs) (Krizhevsky et al., 2012). Therefore, each Residual Unit performs identity mapping of $\mathbf{x}_l$ and nonlinear mapping of $F(\cdot)$. Thanks to this structure, ResNet can compute the output even if we erase $F(\cdot)$ that includes several layers. ResNets also have the structure called *stages* (Greff et al., 2017). A stage has several stacked Residual Units of the same dimensionality for inputs and outputs. Note that the dimensionality can be changed between stages by using down-sampling and increasing channels of convolutions.

In terms of layer pruning for ResNet, Static Layer Pruning is preferable as compared with Dynamic Layer Pruning because it reduces all computation cost. However, previous approaches cannot directly decide the number of layers. In addition, it is difficult for Static Layer Pruning to recover the accuracy because it completely erases layers. Next section introduces our method of Static Layer Pruning, Network Implosion that can directly decide the number of layers and keep the accuracy.

5.4 Proposed method

This section introduces the algorithm of Network Implosion which effectively erases multiple layers and recovers accuracy by employing erasure and retraining scheme. This scheme repeatedly erases layers and retrains the model. In order to realize the erasure and retraining scheme, we need to solve two problems:

(i) Identifying unimportant residual units. ResNet has several important layers whose erasure will dramatically degrade the accuracy. In fact, since ResNet computes new representations when a stage is changed, several layers after the change are important in terms of achieving accuracy (Greff et al., 2017). Therefore, we need to determine the importance of each Residual Unit in order to select those whose erasure will not drastically degrade accuracy.

To solve the problem, we introduce priority, measure of Residual Unit importance.

This priority can be learned from training data in the same way as other parameters in ResNet. We can erase top-k Residual Units in increasing order of priority. In particular, we determine the priority of $F(\cdot)$ in Equation (5.1). Notice that we have $\mathbf{x}_{l+1} = \mathbf{x}_l$ from Equation (5.1) by erasing $F(\cdot)$ as shown in (Veit et al., 2016). This is equivalent to erasing the Residual Unit for Equation (5.1) because input $\mathbf{x}_l$ passes as output to $\mathbf{x}_{l+1}$ without change. In order to determine the importance of $F(\cdot)$, we use the following weighted Residual Unit:

$$\mathbf{x}_{l+1} = \mathbf{x}_l + w_l F(\mathbf{x}_l), \tag{5.2}$$

where w_l is a scalar that can be learned by back propagation in the same way as other ResNet parameters. If w_l is small in terms of absolute value, it scales down the output of $F(\cdot)$. In other words, $F(\cdot)$ has little impact on the result if w_l is small in terms of its absolute value. Therefore, we can select top-k unimportant nonlinear mappings $F(\cdot)$ according to the values of $|w_l|$. However, we should not erase the first Residual Unit in a stage: it is the first Residual Unit after the dimensionality of $\mathbf{x}_l$ changed. Although most Residual Units have outputs of the same dimensionality as inputs in Equation (5.1), the first Residual Unit in a stage changes the dimensionality of the inputs. Greff et al. (2017) suggests that these Residual Units are important with regard to accuracy because they produce new representations in ResNet by changing the dimensionality. Therefore, we use original Residual Units of Equation (5.1) as the first Residual Units in each stage and do not erase them.

(ii) Recovering accuracy for erasing multiple layers. Veit et al. (2016) reports that accuracy decreases significantly when multiple layers are erased. For this problem, we gradually erase Residual Units and retrain the network after each erasure. Specifically, our algorithm has four steps as follows; (i) train the network as usual; (ii) erase a few Residual Units according to $|w_l|$, e.g. $k = 1$; (iii) retrain network until the number of retraining epochs reaches n, the number of epochs for retraining; (iv) repeat (ii) and (iii) until we erase a specified number of layers. Note that (Han et al., 2015) retrains the network one time after erasing parameters in order to maintain the accuracy. However, this approach fails if we erase multiple Residual Units. This is because we erase, at one time, more parameters than the previous method.

In addition, we **retrain the network after the erasure with large learn-**

Algorithm 5.1 Network Implosion.

Input: training set D, initial learning rate η, number of Residual Units L, number of Residual Units to be erased at a time k, total number of Residual Units to be erased L', number of epochs for retraining n

1: Initialize parameters of layers in ResNet and weights $\{w_l : l \in [L]\}$.
2: Train the model by using SGD with η.
3: Set $l' = L$ and $s = 0$.
4: **while** $l' > L'$ **do**
5: $\quad s \leftarrow s + 1$
6: $\quad$ Set $I_s = \emptyset$.
7: $\quad$ Select top-k small elements from $\{|w_l| : l \in [L]\}$ and add the indices into I_s.
8: $\quad$ Erase $w_i F(\mathbf{x}_i)$ in Equation (5.2) where $i \in I_s$.
9: $\quad$ Retrain the model by using SGD with η for n epochs.
10: $\quad l' \leftarrow l' - k$

ing rate. When we erase multiple layers, the structure of the network drastically changes. Thus we need to effectively change the parameters in the remaining layers to efficiently recover the accuracy. To realize this, we set a large learning rate when we retrain the network by using training algorithms such as Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951; Ida et al., 2017). In particular, we reuse the original learning rate of initial network training.

Algorithm 5.1 shows the procedure of Network Implosion. It repeatedly trains and erases Residual Units as explained above. First, we train ResNet with the learning rate η (line 2). Next, we erase top-k nonlinear mappings $F(\cdot)$ according to the importance of $|w_l|$ (lines 5-8). Then, we retrain ResNet with the initial learning rate η (line 9). We repeat the procedure until the accuracy drops or we erase sufficient numbers of layers (lines 4-11).

5.5 Experiments

We performed experiments to evaluate Network Implosion. We first investigated the accuracy of our method, and then computational costs for the inference phase. We implemented our approach in Torch7 (Collobert et al., 2011).

We performed image classification tasks using CIFAR-10, CIFAR-100 (Krizhevsky and Hinton, 2009) and ImageNet dataset (Russakovsky et al., 2015) from ILSVRC2012.

CIFAR-10, CIFAR-100 and ImageNet have 10, 100 and 1000 classes, respectively. The image size is $32 \times 32 \times 3$ for CIFAR-10/100. For ImageNet, we scaled the images $256 \times 256 \times 3$ and took $224 \times 224 \times 3$ single center-crop for training and testing by following (Szegedy et al., 2015). We applied color, scale and aspect ratio augmentation to the images the same as (Krizhevsky et al., 2012; Szegedy et al., 2015).

We implemented ResNet following (He et al., 2016a) as it is used for many image classification tasks. Specifically, each Residual Unit has three convolutional layers with batch normalization and ReLU forming a bottleneck structure. The numbers of stages are three for CIFAR-10/100, and four for ImageNet. As a result, the number of layers is 56 for CIFAR-10/100, and 50 for ImageNet, the same as (He et al., 2016b). We used projection shortcuts (He et al., 2016a) for increasing dimensions; this is used when stages are changed. The other hyper parameters were also set according to (He et al., 2016a) which is widely used in the deep learning community; the number of epochs was 200; the training algorithm was SGD with momentum; the momentum was 0.9; the initial learning rate was 0.1; the learning rate was divided by 10 at 81 and 122 epochs for the CIFAR-10 and CIFAR-100 datasets. For ImageNet, we decayed the leaning rate by multiplying the learning rate by 0.1 at every 30 epochs. The mini-batch size was 128 for CIFAR-10/100, and 512 for ImageNet. The weight decay was 0.0001. The parameters in each layer were initialized as in (He et al., 2015), a standard method for deep neural networks with ReLU activations.

For comparison, we also used the teacher-student training regime based on Knowledge Distillation (Hinton et al., 2015; Ba and Caruana, 2014). The teacher-student training regime can reduce the number of layers without additional computation costs for the inference while previous dynamic layer pruning cannot as described in the section of related work. In addition, teacher-student training regime can directly detemines the number of layers while previous static layer pruning cannot. We used 56 and 50 layered ResNets as the teacher networks for CIFAR-10/100 and ImageNet, respectively. The temperature was 4, and we used 0.9 for a tunable parameter to balance cross entropies in CIFAR-10/100 as a result of grid search (see (Hinton et al., 2015) for a description of these hyper parameters). In ImageNet, the temperature was 1 and we used 0.5 for the tunable parameter.

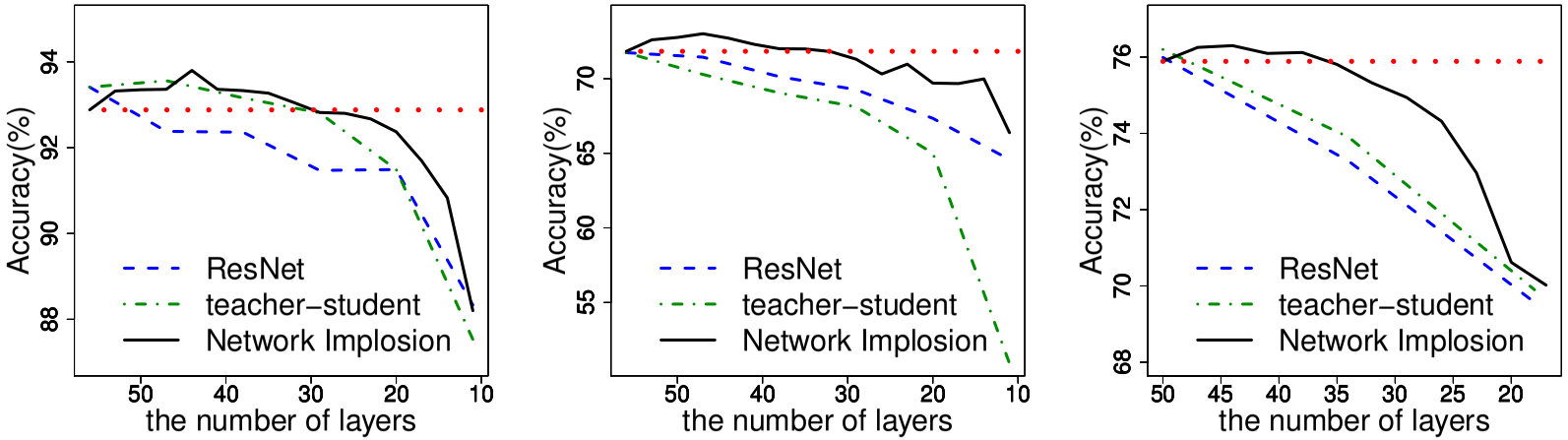
In our approach, we used Residual Units of Equation (5.2) when $\mathbf{x}_{l+1}$ had the same dimensionality as $\mathbf{x}_l$. When the dimensionality of $\mathbf{x}_{l+1}$ differed from $\mathbf{x}_l$, we used

the original Residual Units of Equation (5.1) and did not erase them. This is because such Residual Units are important for generating new representations as described in the previous sections. We erased one Residual Unit (k=1) after each training or retraining cycle. Since each Residual Unit has three convolution layers, we can erase three layers by erasing a Residual Unit. In the retraining phase, we used the initial learning rate of 0.1. The number of epochs was 60. We divided the learning rate by 10 at 20 and 40 epochs.

5.5.1 Accuracy

We evaluated the validation accuracies of our approach, teacher-student training regime, and original ResNet. For CIFAR-10/100, we reduced the number of layers from 56 to 11. For ImageNet, we trained 50, 34 and 18 layer models for original ResNet and teacher-student training regime while our method reduced the layers from 50 to 17.

Figure 5.1 shows the experimental results. The red dotted lines in the figure are initial accuracies of Network Implosion; these are accuracies of 56 and 50 layer models for CIFAR-10/100 and ImageNet, respectively. Although Network Implosion erases layers from the model, it yielded accuracies above the red dotted lines by erasing layers. Finally, we could reduce the number of layers to 32, 35 and 38 for CIFAR-10, CIFAR-100 and ImageNet without accuracy loss, respectively. These numbers correspond to crossover points of the red and black lines in Figure 5.1. In other words, we could reduce the number of layers by 42.86%, 37.50% and 24.00% for CIFAR-10, CIFAR-100 and ImageNet, respectively. Notice that when we simply reduced the number of layers in original ResNet, the accuracies fell even though the number of training epochs was the same as our method (blue dashed lines in the figures). Although the teacher-student training regime achieves comparable accuracies to our method for CIFAR-10, it rapidly degrades the accuracies as more layers were eliminated for CIFAR-100 and ImageNet (green dot-dash lines in the figures). In particular, it drastically degraded the accuracy at 34 layers whereas our algorithm kept the accuracy high for ImageNet. These results reveal that Network Implosion is effective in reducing the number of layers even if we use real-world datasets such as ImageNet.



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet

Figure 5.1: Accuracies achieved for CIFAR-10, CIFAR-100, and ImageNet. The red dotted lines represent accuracies for initial models in our approach: Network Implosion. It achieves higher accuracies than baselines even though it erases many layers.

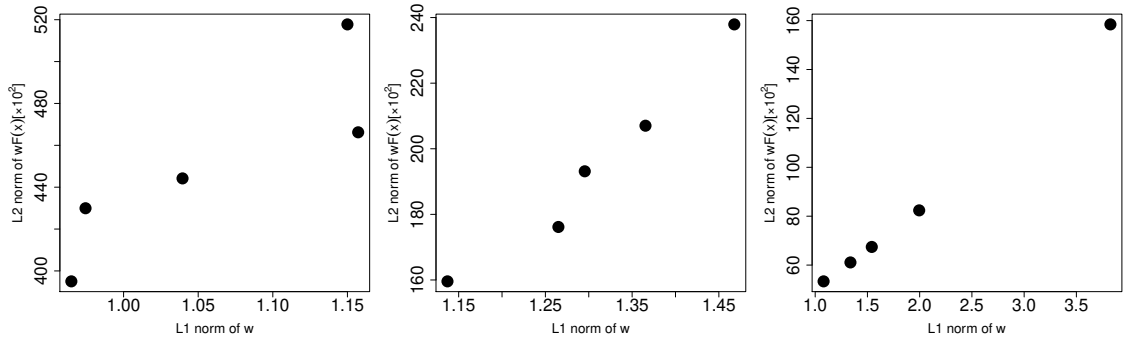
5.5.2 Computation Costs

We evaluated the computation costs for the inference phase: the number of MAC (multiply-accumulate) operations, the processing times of forward and backward propagations, and the number of parameters. MAC is the main operation of deep neural networks and is used in convolution and fully-connected layers. As the processing times, we averaged 100 runs for forward and backward propagation. In addition, we counted the number of parameters to be learned for evaluating model sizes. We used the same setting as the previous section for training. By using Network Implosion, we reduced 56-layered models to 32 and 35 layers for CIFAR-10 and CIFAR-100, respectively. For ImageNet, we reduced the 50-layered model to 38 layers. These models are the smallest models with no drop in accuracy as described in Figure 5.1.

Table 5.1 shows the results. The table shows that the numbers of MACs are reduced to 60.93 %, 62.89 % and 78.59 % of baselines for CIFAR-10, CIFAR-100, and ImageNet, respectively. In proportion to the number of MACs, we could reduce the processing times of forward propagation to 60.23 %, 70.13 % and 76.69 % of baselines for each dataset. For backward propagation, which is used for fine-tuning in Transfer Learning (Yosinski et al., 2014), we could achieve 59.71 %, 60.44 % and 78.19 % of the processing times for the respective datasets. In terms of model size, our approach reduced the number of parameters to 69.82 %, 90.50 % and 93.15 % of baselines for CIFAR-10, CIFAR-100 and ImageNet with no drop in accuracy, respectively. The results reveal that our approach erases layers without additional computation costs or drop in accuracy for the inference phase.

Table 5.1: The computation costs for the inference phase after erasing layers without accuracy loss. 56 and 50 layered models are original models for CIFAR-10/100 and ImageNet, respectively. Our method reduces all computation cost without accuracy loss.

dataset	# of layers	accuracy (%)	# of multiply-accumurate operations (MACs)	forward (msec)	backward (msec)	# of parameters
CIFAR-10	56	92.88	8.19×10^7	6.584	12.93	585.9K
	32	**93.05**	$\mathbf{4.99 \times 10^7}$	**3.970**	**7.721**	**409.1K**
CIFAR-100	56	71.83	8.65×10^7	6.203	13.36	613.6K
	35	**71.99**	$\mathbf{5.44 \times 10^7}$	**4.350**	**8.075**	**555.3K**
ImageNet	50	75.89	4.11×10^9	29.95	59.51	25.55M
	38	**76.12**	$\mathbf{3.23 \times 10^9}$	**22.97**	**46.53**	**23.80M**



(a) Stage 1 (b) Stage 2 (c) Stage 3

Figure 5.2: Scatter plot of the absolute values of priorities w_l and the means of the magnitudes (l_2 norm) of the scaled output $w_l F(\mathbf{x}_l)$ for each stage of 56-layered ResNet trained with CIFAR-10. We can see that when the priority term takes the small value, the output also takes the small value.

5.5.3 Scale-up/down Effect of Priority Term

In the proposed method, the absolute value of w_l is used as a measure of the importance of Residual Units. In other words, our idea assumes that the output of $F(\mathbf{x}_l)$ is scaled down when the absolute value of w_l is small. As a result, the magnitude of the output of $w_l F(\mathbf{x}_l)$ is small, and its impact on the output of the model is reduced. To confirm this assumption, we visualized the relationship between the absolute values of w_l and the means of the magnitudes of $w_l F(\mathbf{x}_l)$ (l_2 norm) on the validation dataset for 56-layer ResNet trained with CIFAR-10 for each stage. Figure 5.2 shows the results. From this figure, we can see that when the absolute values of w_l are small, the l_2 norms of $w_l F(\mathbf{x}_l)$ are also small. The results suggest that $F(\mathbf{x}_l)$ is scaled down or up by w_l, and the absolute value of w_l can be empirically used as the importance of Residual Unit.

5.6 Summary

We proposed Network Implosion that can erase multiple layers from ResNets with no loss of accuracy. It offers high accuracy by using priority to select which Residual Units to erase; the remaining units are retrained with large learning rate. We evaluated our approach on CIFAR-10, CIFAR-100 and ImageNet. The results show

that Network Implosion effectively reduces the number of layers without degrading the accuracy even on real-world datasets such as ImageNet.

Conclusion and Future Direction

Although machine learning has become a part of our life, the computation cost has increased due to the increase in model sizes. The theme of this dissertation is to propose fast algorithms for such large models. In this dissertation, we categorized the large models into wide models (Part I) and deep models (Part II), and proposed fast algorithms that utilized the characteristics for each category. In this chapter, we summarize the works for Parts I and II, and point out some potential research directions.

Conclusion

Part I: Fast algorithms for wide models.

We proposed fast algorithms for wide models that take the form of long vectors (Chapter 2) and large matrices (Chapter 3) as their parameters. The key to the proposed algorithms is to introduce sparsity and compute upper and lower bounds of the condition scores for the parameters to be zeros. The proposed algorithms have the following attractive properties by utilizing the upper/lower bound techniques:

- **High speed:** Our algorithms skip unnecessary computations and intensively update important parameters by utilizing upper bounds and lower bounds, respectively. Experiments demonstrated that our methods are faster than the original methods by up to tens of times.
- **High accuracy:** Our algorithms are guaranteed to achieve the same objective values as that of the original algorithm. This is because the skips of computations in our algorithms are safe, and the optimization

problems are convex optimizations. Experiments showed that our method achieved the same objective values as that of the original methods.

- **No additional hyperparameters:** We designed our algorithms to have no additional hyperparameters. Thus our algorithms are practitioner-friendly as they do not increase the tuning costs for hyperparameters.

Besides, we also showed that it is relatively easy to extend our methods to various types of data such as overlapping features, graph-structured features, and tensors. Therefore, our algorithms would be useful tools in dealing with a wide variety of data.

Part II: Fast algorithms for deep models.

We focus on deep neural networks that are the main deep models in the field of recent artificial intelligence. Since it is crucial problems that deep neural networks take long processing times for phases of training (Chapter 4) and inference (Chapter 5), we proposed fast algorithms for each phase as follows:

Training phase. For the training phase, we proposed the adaptive learning rate on the basis of the covariance matrix based preconditioning (Chapter 4). Although our method does not reduce the time complexity of the training, it could effectively reduce the training loss: it can reach the same accuracy as that of existing methods in less time. The characteristic point of our algorithm is to effectively handle the noise present in the gradients on the basis of covariance matrix based preconditioning. Thanks to this property, it can effectively train models even if we use small mini-batch sizes which incur noisy gradients. Since it can be used for various types of deep neural networks such as MLP, RNN, and CNN, it would be helpful to accelerate the trial-and-error of the training for a wide range of applications.

Inference phase. For the inference phase, we focused on residual networks which is a popular model in the field of computer vision (Chapter 5). As the processing time of the inference increases due to a large number of layers, we proposed the method of erasing unimportant layers. Surprisingly, it could keep accuracy by retraining the model with a large learning

rate even if several layers were erased. Namely, we can accelerate the inference without degrading accuracy. The algorithm would reduce the running costs of machine learning applications after service deployment. In addition, our approach can be used for models that have identity maps like ResNet. As various models with identity maps have appeared, e.g Transformers in natural language processing (So et al., 2019), our algorithm would be widely used for various applications other than computer vision.

This dissertation proposed fast algorithms for wide and deep models as we summarized the above. As the amount of data continues to increase, we believe that our algorithms allow us to handle larger data.

Future Direction

Finally, we list some future directions that sound interesting below:

- **Extension to other types of regularizations:** In Part I, we proposed the upper/lower bound technique for l_2 norm regularization. An interesting research direction is whether this technique can be extended to other types of regularizations. In particular, the speeding-up of optimizations with non-convex regularizations is a challenging problem in the community of sparse optimization although the non-convex regularizations have statistically good properties (Fan and Li, 2001). This is because most existing methods utilize convexity to accelerate the optimizations. Since our upper/lower bound technique does not assume the convexity, it will be applicable to non-convex regularizations.

- **Speed-up without warm start strategy:** In Part I, our methods and most existing methods utilize warm start strategy with the sequential rule. Although the sequential rule is similar to a grid search for regularization constants, we cannot parallelize the search due to the warm start strategy. Therefore, it would be preferable to propose fast methods without warm start strategy for practitioners.

- **Theoretical analysis about depth of deep models:** In Chapter 5, we proposed the method that erases unimportant layers to accelerate the inference. Surprisingly, we empirically show that it could erase several layers without degrading accuracy even on a real world dataset. The result suggests the next research question: what is the minimum number of layers needed to produce the highest accuracy in theory? Although our method requires additional computation cost due to the erasure and retraining scheme, we may save the cost if the above question is answered.

- **Upper/lower bound technique for deep models:** In Chapter 4, we proposed an adaptive learning rate on the basis of covariance matrix based preconditioning. However, we found the important observation from the experimental results of Chapter 5: we can erase several layers without degrading accuracy even on a real-world dataset. In other words, several layers turn to be unnecessary layers during the optimization. For such a situation, we may use the upper/lower bound technique proposed in Part I because it can skip unnecessary computations during the optimizations. We may accelerate the training of ResNets by utilizing the upper/lower bound technique in addition to covariance matrix based preconditioning.

Publications

Major Publications

International Conference

1. Yasutoshi Ida, Yasuhiro Fujiwara, and Sotetsu Iwamura. Adaptive Learning Rate via Covariance Matrix Based Preconditioning. In *Proceedings of International Conference on Artificial Intelligence (IJCAI)*, 1923–1929, 2017.

2. Yasutoshi Ida and Yasuhiro Fujiwara. Network Implosion: Effective Model Compression for ResNets via Static Layer Pruning and Retraining. In *International Joint Conference on Neural Networks (IJCNN)*, 1–8, 2019.

3. Yasutoshi Ida, Yasuhiro Fujiwara, and Hisashi Kashima. Fast Sparse Group Lasso. In *Proceedings of Conference on Neural Information Processing Systems (NeurIPS)*, 1700–1708, 2019.

4. Yasutoshi Ida and Yasuhiro Fujiwara. Improving Generalization Performance of Adaptive Learning Rate by Switching from Block Diagonal Matrix Preconditioning to SGD. In *International Joint Conference on Neural Networks (IJCNN)*, 1–8, 2020.

5. Yasutoshi Ida, Sekitoshi Kanai, Yasuhiro Fujiwara, Tomoharu Iwata, and Hisashi Kashima. Fast Deterministic CUR Matrix Decomposition with Accuracy Assurance. In *Proceedings of International Conference on Machine Learning (ICML)*, 4594–4603, 2020.

Journals

1. 井田 安俊, 藤原 靖宏. “層の削除と再学習によるResNetのモデル圧縮”. 人工知能学会論文誌, 35(3), C-JA3_1-10, 2020.

2. 井田 安俊, 藤原 靖宏, 鹿島 久嗣. “Sparse Group Lasso のための高速な Block Coordinate Descent”. 人工知能学会論文誌, 36(1), A-JB1_1-11, 2021.

Others

International Conference

1. Tomohiko Suzuki, Takuma Nakamura, Yasutoshi Ida, and Takashi Matsumoto. Handling Incomplete Matrix Data via Continuous-Valued Infinite Relational Model. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2153–2156, 2012.

2. Yasutoshi Ida, Takuma Nakamura, and Takashi Matsumoto. Domain-Dependent /Independent Topic Switching Model for Online Reviews with Numerical Ratings. In *Conference on Information and Knowledge Management (CIKM)*, 229–238, 2013.

3. Hiroki Miyashita, Takuma Nakamura, Yasutoshi Ida, Takashi Matsumoto, and Takashi Kaburagi. Nonparametric Bayes-based Heterogeneous ”Drosophila melanogaster” Gene Regulatory Network Inference: T-Process Regression. In *International Conference on Artificial Intelligence and Applications*, 2013.

4. Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Yasutoshi Ida, and Machiko Toyoda. Adaptive Message Update for Fast Affinity Propagation. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 309–318, 2015.

5. Yasuhiro Fujiwara, Yasutoshi Ida, Hiroaki Shiokawa, and Sotetsu Iwamura. Fast Lasso Algorithm via Selective Coordinate Descent. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1561–1567, 2016.

6. Yasuhiro Fujiwara, Junya Arai, Sekitoshi Kanai, Yasutoshi Ida, and Naonori Ueda. Adaptive Data Pruning for Support Vector Machines. In IEEE BigData, 683-692, 2018.

7. Yasuhiro Fujiwara, Yasutoshi Ida, Sekitoshi Kanai, Atsutoshi Kumagai, Junya Arai, and Naonori Ueda. Fast Random Forest Algorithm via Incremental Upper Bound. In *Conference on Information and Knowledge Management (CIKM)*, 2205–2208, 2019.

8. Yasuhiro Fujiwara, Sekitoshi Kanai, Junya Arai, Yasutoshi Ida, and Naonori Ueda. Efficient Data Point Pruning for One-Class SVM. In *AAAI Conference on Artificial Intelligence (AAAI)*, 3590–3597, 2019.

9. Yasuhiro Fujiwara, Atsutoshi Kumagai, Sekitoshi Kanai, Yasutoshi Ida, and Naonori Ueda. Efficient Algorithm for the b-Matching Graph. In *Conference on Knowledge Discovery and Data Mining (KDD)*,187–197, 2020.

10. Sekitoshi Kanai, Yasutoshi Ida, Yasuhiro Fujiwara, Masanori Yamada, and Shuichi Adachi. Absum: Simple Regularization Method for Reducing Structural Sensitivity of Convolutional Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 4394–4403, 2020.

Domestic Conference

1. 宮下 弘樹, 中村 拓磨, 井田 安俊, 鈴木知彦, 松本 隆, 鏑木 崇史. “ノンパラメトリックベイジアン T 過程アルゴリズムによる時間的構造変化を考慮した遺伝子発現ネットワーク推定”, バイオ情報学研究会, 32(15), 1–6, 2012.

2. 井田 安俊, 藤原 靖宏, 岩村 相哲. “ディープニューラルネットワークのための勾配上のノイズを考慮した効率的な確率的勾配降下法”, 情報論的学習理論と機械学習 (IBISML) 研究会, 116(209), 93–94, 2016.

3. 藤原 靖宏, 井田 安俊, 塩川 浩昭, 岩村 相哲. “選択的座標降下法による Lasso の高速化”, 人工知能学会全国大会論文集, JSAI2016, 1F44, 2016.

4. 大屋 優, 井田 安俊, 藤原 靖宏, 岩村 相哲. “正則化による深層学習の重み量子化手法の検討”, パターン認識・メディア理解研究会 (PRMU), 117(210), 51–52, 2017.

5. 大屋 優, 井田 安俊, 藤原 靖宏, 岩村 相哲. “バイナリニューラルネットにおける非活性化手法の検討”, パターン認識・メディア理解研究会 (PRMU), 117(238), 119–120, 2017.

Journals

1. Yukikatsu Fukuda, Yasutoshi Ida, Takashi Matsumoto, Naohiro Takemura, and Kaoru Sakatani. A Bayesian Algorithm for Anxiety Index Prediction Based on CerebralBlood Oxygenation in the Prefrontal Cortex Measured by Near Infrared Spectroscopy. In *IEEE Journal of Translational Engineering in Health and Medicine*, 2014.

2. Yasuhiro Fujiwara, Yasutoshi Ida, Junya Arai, Mai Nishimura, and Sotetsu Iwamura. Fast Algorithm for the Lasso based L1-Graph Construction. *Proceedings of the Very Large Data Bases (PVLDB)*, 10(3), 229–240, 2016.

Technical Reports

1. 井田 安俊, 金井 関利, 藤原 靖宏, 八木 哲志, 飯田 恭弘. “深層学習のための先進的な学習技術”. NTT技術ジャーナル, 30(6), 30–33, 2018.

2. Yasutoshi Ida, Sekitoshi Kanai, Yasuhiro Fujiwara, Satoshi Yagi, and Yasuhiro Iida. Advanced Learning Technologies for Deep Learning. NTT Technical Review, 16(8), 37–41, 2018.

Workshops

1. 井田 安俊, 鈴木 知彦, 松本 隆. “半教師あり学習を用いたノンパラメトリックトピックモデルによるweb文書の評判分析”. 第14回情報論的学習理論ワークショップ (IBIS2011), 2011年11月.

2. 鈴木 知彦, 井田 安俊, 松本 隆. “文書情報を考慮した無限関係モデルによるネットワーク推定”. 第14回情報論的学習理論ワークショップ (IBIS2011), 2011年11月.

3. 小笠原 光貴, 井田 安俊, 横井 創磨, 松本 隆. “文書への自動的なタグ付けを実現する、高速な手法に関する検討”. 第16回情報論的学習理論ワークショップ (IBIS2013), 2013年11月.

4. 横井 創磨, 井田 安俊, 小笠原 光貴, 松本 隆. “構文構造を考慮した特徴量を用いたトピックモデルによる評判分析”. 第16回情報論的学習理論ワークショップ (IBIS2013), 2013年11月.

5. Yasutoshi Ida, Takuma Nakamura, and Takashi Matsumoto. Label-Related/ Unrelated Topic Switching Model: A Partially Labeled Topic Model Handling Infinite Label-unrelated Topics. Recent Advances in Computer Vision and Pattern Recognition, 2013.

6. 井田安俊, 藤原靖宏, 鹿島久嗣. “Fast Sparse Group Lasso”. 第22回情報論的学習理論ワークショップ (IBIS2019), 2019年11月.

7. 井田 安俊, 金井 関利, 藤原 靖宏, 岩田 具治, 竹内 孝, 鹿島 久嗣. “スパース正則化に基づいたCUR行列分解の高速化”. 第23回情報論的学習理論ワークショップ (IBIS2020), 2020年11月.

Patents

1. 井田安俊, 藤原靖宏. “学習装置, 学習方法及び学習プログラム”. 出願番号: 特願2018-513164, 出願日: 2017年4月14日, 公報番号: 特許第6417075号, 公報日: 2018年10月31日, 特許番号: 特許6417075, 登録日: 2018年10月12日.

2. 井田安俊. “学習装置, 学習方法および学習プログラム”. 出願番号: 特願2018-73498, 出願日: 2018年4月5日, 公報番号: 特開2019-185275, 公報日: 2019年10月24日.

3. 井田安俊, 藤原靖宏. “データ分析装置, データ分析方法及びプログラム”. 出願番号: 特願2019-75952, 出願日: 2019年4月11日. 公報番号: 特開2020-173674, 公報日: 2020年10月22日.

4. 井田安俊. “データ処理方法, データ処理装置及びデータ処理プログラム”. 出願番号: 特願2020-18013, 出願日: 2020年4月27日.

5. Yasutoshi Ida. LEARNING DEVICE, LEARNING METHOD, AND LEARNING PROGRAM. Appl. No.: WO2017JP015337W, Pub. No.: WO2017183587A.

6. Yasutoshi Ida. LEARNING APPARATUS, LEARNING METHOD, AND RECORDING MEDIUM. Appl. No.: US16/092135, Pub. No.: US2019/0156240.

7. Yasutoshi Ida. LEARNING DEVICE, LEARNING METHOD, AND LEARNING PROGRAM. Appl. No.: EP17785925A, Pub. No.: EP3432230A.

8. Yasutoshi Ida. LEARNING DEVICE, LEARNING METHOD, AND LEARNING PROGRAM. Appl. No.: WO2019JP015040W, Pub. No.: WO2019194299A.

9. Yasutoshi Ida. DATA ANALYSIS DEVICE, DATA ANALYSIS METHOD, AND DATA ANALYSIS PROGRAM. Appl. No.: WO2020JP013688W, Pub. No.: WO2020209086A.

Bibliography

1000 Genomes Project Consortium and others. A Global Reference for Human Genetic Variation. *Nature*, 526(7571):68–74, 2015.

T. Akiba, S. Suzuki, and K. Fukuda. Extremely large minibatch SGD: training ResNet-50 on ImageNet in 15 minutes. *CoRR*, abs/1711.04325, 2017.

S. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10 (2):251–276, 1998.

J. Ba and R. Caruana. Do Deep Nets Really Need to be Deep? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, pages 2654–2662, 2014.

S. Bianco, R. Cadène, L. Celona, and P. Napoletano. Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access*, 6:64270–64277, 2018.

J. Bien, Y. Xu, and M. W. Mahoney. CUR from a Sparse Optimization Viewpoint. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 23, pages 217–225, 2010.

A. Bonnefoy, V. Emiya, L. Ralaivola, and R. Gribonval. A Dynamic Screening Principle for the Lasso. In *European Signal Processing Conference*, pages 6–10, 2014.

A. Bonnefoy, V. Emiya, L. Ralaivola, and R. Gribonval. Dynamic Screening: Accelerating First-Order Algorithms for the Lasso and Group-lasso. *IEEE Trans. Signal Processing*, 63(19):5121–5132, 2015.

C. Chang and C. Lin. LIBSVM: A Library for Support Vector Machines. *ACM TIST*, 2(3):27:1–27:27, 2011.

R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, Workshop in Neural Information Processing Systems*, 2011.

Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, pages 2933–2941, 2014.

O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal Distributed Online Prediction using Mini-batches. *Journal of Machine Learning Research (JMLR)*, 13:165–202, 2012.

P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo Algorithms for Matrices III: Computing a Compressed Approximate Matrix Decomposition. *SIAM J. Comput.*, 36(1):184–206, 2006.

P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Relative-Error CUR Matrix Decompositions. *SIAM J. Matrix Analysis Applications*, 30(2):844–881, 2008.

J. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990.

J. Fan and R. Li. Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise Coordinate Optimization. *Ann. Appl. Stat.*, 1(2):302–332, 12 2007.

J. Friedman, T. Hastie, and R. Tibshirani. A Note on The Group Lasso and a Sparse Group Lasso. *arXiv preprint arXiv:1001.0736*, 2010.

Y. Fujiwara, Y. Ida, J. Arai, M. Nishimura, and S. Iwamura. Fast Algorithm for the Lasso based L1-Graph Construction. *In Proceedings of the Very Large Database Endowment (PVLDB)*, 10(3):229–240, 2016a.

Y. Fujiwara, Y. Ida, H. Shiokawa, and S. Iwamura. Fast Lasso Algorithm via Selective Coordinate Descent. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 1561–1567, 2016b.

L. E. Ghaoui, V. Viallon, and T. Rabbani. Safe Feature Elimination for the Lasso and Sparse Supervised Learning Problems. *Pacific Journal of Optimization*, 8(4): 667–698, 2012.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

K. Greff, R. K. Srivastava, and J. Schmidhuber. Highway and Residual Networks Learn Unrolled Iterative Estimation. *International Conference on Learning Representations (ICLR)*, 2017.

N. Halko, P.-G. Martinsson, and J. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM review*, 53(2):217–288, 2011.

S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, pages 1135–1143, 2015.

T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. In *European Conference on Computer Vision (ECCV)*, pages 630–645, 2016a.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE Computer Society, 2016b.

G. E. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *CoRR*, abs/1503.02531, 2015.

J. Huang, P. Breheny, and S. Ma. A Selective Review of Group Selection in High-Dimensional Models. *Statistical Science*, 27(4):481–499, 2012.

Z. Huang and N. Wang. Data-Driven Sparse Structure Selection for Deep Neural Networks. In *European Conference on Computer Vision (ECCV)*, pages 317–334, 2018.

Y. Ida, Y. Fujiwara, and S. Iwamura. Adaptive Learning Rate via Covariance Matrix Based Preconditioning for Deep Neural Networks. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1923–1929, 2017.

Y. Ida, Y. Fujiwara, and H. Kashima. Fast Sparse Group Lasso. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 1700–1708, 2019.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 37, pages 448–456, 2015.

L. Jacob, G. Obozinski, and J. Vert. Group Lasso with Overlap and Graph Lasso. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 433–440, 2009.

R. Jenatton. *Structured Sparsity-Inducing Norms: Statistical and Algorithmic Properties with Applications to Neuroimaging*. PhD thesis, 2011.

T. B. Johnson and C. Guestrin. Unified Methods for Exploiting Piecewise Linear Structure in Convex Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 4754–4762, 2016.

T. B. Johnson and C. Guestrin. StingyCD: Safely Avoiding Wasteful Updates in Coordinate Descent. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 70, pages 1752–1760, 2017.

A. Karpathy, J. Johnson, and F.-F. Li. Visualizing and Understanding Recurrent Networks. *arXiv preprint arXiv:1506.02078*, 2015.

D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *International Conference in Learning Representations (ICLR)*, 2014.

A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, pages 1097–1105, 2012.

G. Larsson, M. Maire, and G. Shakhnarovich. FractalNet: Ultra-deep neural networks without residuals. In *ICLR*, 2017.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu. Feature Selection: A Data Perspective. *ACM Comput. Surv.*, 50(6), Dec. 2017.

L. W. Mackey, A. Talwalkar, and M. I. Jordan. Divide-and-Conquer Matrix Factorization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1134–1142, 2011.

M. W. Mahoney and P. Drineas. CUR Matrix Decompositions for Improved Data Analysis. *Proc. Natl. Acad. Sci. U.S.A.*, 106(3):697–702, 2009.

M. W. Mahoney, M. Maggioni, and P. Drineas. Tensor-CUR Decompositions for Tensor-based Data. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 327–336, 2006.

J. Mairal, R. Jenatton, G. Obozinski, and F. R. Bach. Convex and Network Flow Optimization for Structured Sparsity. *Journal of Machine Learning Research (JMLR)*, 12:2681–2720, 2011.

N. Meinshausen and P. Buhlmann. High-dimensional Graphs and Variable Selection with the Lasso. *Annals of Statistics*, 34:1436–1462, 2006.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1st edition, 1997.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2nd edition, 2018.

D. Nam and S.-Y. Kim. Gene-set Approach for Expression Pattern Analysis. *Brief. Bioinforma.*, 9(3):189–197, 2008.

E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. GAP Safe Screening Rules for Sparse Multi-task and Multi-class models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 811–819, 2015.

E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. GAP Safe Screening Rules for Sparse-Group Lasso. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 388–396, 2016.

E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. Gap Safe Screening Rules for Sparsity Enforcing Penalties. *Journal of Machine Learning Research (JMLR)*, 18(1):4671–4703, 2017.

A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding Gradient Noise Improves Learning for Very Deep Networks. *arXiv preprint arXiv:1511.06807*, 2015.

G. Obozinski, L. Jacob, and J.-P. Vert. Group Lasso with Overlaps: the Latent Group Lasso approach. Research report, 2011.

D. S. Papailiopoulos, A. Kyrillidis, and C. Boutsidis. Provable Deterministic Leverage Score Sampling. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 997–1006, 2014.

R. Pascanu, T. Mikolov, and Y. Bengio. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1310–1318, 2013.

S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C.-H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. Mesirov, T. Poggio, W. Gerald, M. Loda, and E. Lander. Multiclass Cancer Diagnosis using Tumor Gene Expression Signatures. *Proceedings of the National Academy of Sciences*, 98, 01 2002.

H. Robbins and S. Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

V. Roth and B. Fischer. The group-Lasso for Generalized Linear Models: Uniqueness of Solutions and Efficient Algorithms. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 848–855, 2008.

N. Roux, P.-A. Manzagol, and Y. Bengio. Topmoumoute Online Natural Gradient Algorithm. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 849–856, 2008.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

M. Schuster. Speech Recognition for Mobile Devices at Google. In *Pacific Rim International Conference on Artificial Intelligence*, pages 8–10, 2010.

F. Seide, G. Li, and D. Yu. Conversational Speech Transcription Using Context-Dependent Deep Neural Networks. In *INTERSPEECH*, pages 437–440. ISCA, 2011.

P. Sermanet, S. Chintala, and Y. LeCun. Convolutional Neural Networks Applied to House Numbers Digit Classification. In *International Conference on Pattern Recognition (ICPR)*, pages 3288–3291, 2012.

S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014.

N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. A Sparse-Group Lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013.

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556, 2014.

D. R. So, Q. V. Le, and C. Liang. The Evolved Transformer. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 97, pages 5877–5886, 2019.

S. Sra, S. Nowozin, and S. Wright. *Optimization for Machine Learning*. Mit Press, 2012.

J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is More: Compact Matrix Decomposition for Large Sparse Graphs. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, pages 366–377, 2007.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, pages 3104–3112, 2014.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

R. Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288, 1996.

R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani. Strong Rules for Discarding Predictors in Lasso-type Problems. *Journal of the Royal Statistical Society Series B*, 74(2):245–266, 2012.

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of its Recent Magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

H. Tong, S. Papadimitriou, J. Sun, P. S. Yu, and C. Faloutsos. Colibri: Fast Mining of Large Static and Dynamic Graphs. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 686–694, 2008.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

A. Veit and S. J. Belongie. Convolutional Networks with Adaptive Computation Graphs. In *European Conference on Computer Vision (ECCV)*, pages 3–18, 2018.

A. Veit, M. J. Wilber, and S. J. Belongie. Residual Networks Behave Like Ensembles of Relatively Shallow Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 550–558, 2016.

J. Wang and J. Ye. Two-Layer Feature Reduction for Sparse-Group Lasso via Decomposition of Convex Sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2132–2140, 2014.

J. Wang, J. Zhou, P. Wonka, and J. Ye. Lasso Screening Rules via Dual Polytope Projection. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1070–1078, 2013.

W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning Structured Sparsity in Deep Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 2074–2082, 2016.

J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized Convolutional Neural Networks for Mobile Devices. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4820–4828, 2016.

T. T. Wu, Y. F. Chen, T. Hastie, E. Sobel, and K. Lange. Genome-Wide Association Analysis by Lasso Penalized Logistic Regression. *Bioinformatics*, 25(6):714–721, Mar. 2009.

Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. S. Feris. BlockDrop: Dynamic Inference Paths in Residual Networks. In *Computer Vision and Pattern Recognition (CVPR)*, pages 8817–8826, 2018.

X. Xiao, L. Jin, Y. Yang, W. Yang, J. Sun, and T. Chang. Building Fast and Compact Convolutional Neural Networks for Offline Handwritten Chinese Character Recognition. *Pattern Recognition*, 72:72–81, 2017.

J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How Transferable are Features in Deep Neural Networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, pages 3320–3328, 2014.

X. Yu, Z. Yu, and S. Ramalingam. Learning Strict Identity Mappings in Deep Residual Networks. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4432–4440, 2018.

M. Yuan and Y. Lin. Model Selection and Estimation in Regression with Grouped Variables. *Journal of the Royal Statistical Society, Series B*, 68(1):49–67, 2006.

S. Zagoruyko and N. Komodakis. Wide Residual Networks. In *British Machine Vision Conference(BMVC)*, pages 87.1–87.12, 2016.

M. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv preprint arXiv:1212.5701*, 2012.

P. Zhu, W. Zuo, L. Zhang, Q. Hu, and S. C. K. Shiu. Unsupervised Feature Selection by Regularized Self-Representation. *Pattern Recognit.*, 48(2):438–446, 2015.