FFTLasso: Large-Scale LASSO in the Fourier Domain

Adel Bibi, Hani Itani, Bernard Ghanem
King Abdullah University of Science and Technology (KAUST), Saudi Arabia
adel.bibi@kaust.edu.sa, hmi10@mail.aub.edu, bernard.ghanem@kaust.edu.sa

Abstract

In this paper, we revisit the LASSO sparse representation problem, which has been studied and used in a variety of different areas, ranging from signal processing and information theory to computer vision and machine learning. In the vision community, it found its way into many important applications, including face recognition, tracking, super resolution, image denoising, to name a few. Despite advances in efficient sparse algorithms, solving large-scale LASSO problems remains a challenge. To circumvent this difficulty, people tend to downsample and subsample the problem (e.g. via dimensionality reduction) to maintain a manageable sized LASSO, which usually comes at the cost of losing solution accuracy. This paper proposes a novel circulant reformulation of the LASSO that lifts the problem to a higher dimension, where ADMM can be efficiently applied to its dual form. Because of this lifting, all optimization variables are updated using only basic element-wise operations, the most computationally expensive of which is a 1D FFT. In this way, there is no need for a linear system solver nor matrix-vector multiplication. Since all operations in our FFTLasso method are element-wise, the subproblems are completely independent and can be trivially parallelized (e.g. on a GPU). The attractive computational properties of FFTLasso are verified by extensive experiments on synthetic and real data and on the face recognition task. They demonstrate that FFTLasso scales much more effectively than a state-of-the-art solver.

1. Introduction

In this paper, we are interested in efficiently solving the popular LASSO problem, defined as follows:

$$\min_{\mathbf{c}} \underbrace{\|\mathbf{A}\mathbf{c}-\mathbf{b}\|_2^2}_{f(\mathbf{c})} + \lambda \underbrace{\|\mathbf{c}\|_1}_{g(\mathbf{c})}, \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the sparsifying dictionary and λ is a positive parameter that trades off between sparsity and data fitting fidelity. Problem (1) is strongly convex but non-smooth. Problem (1) describes many applications of interest in both computer vision and machine learning, including (but not limited to) face recognition [32, 28, 33], face alignment [25, 24], tracking [20, 35, 37, 38, 15, 36, 3], super resolution [34, 27], convolutional sparse coding [6, 14, 17], image inpainting [30], sparse subspace clustering [11, 10], image debluring [2, 16, 26], just to name a few.

In many applications (*e.g.* face recognition and face alignment), it is important to solve problem (1) for a very large overcomplete dictionary $\mathbf{A}$ (*e.g.* when $m, n \gg 10^5$). In face recognition, face alignment, and in tracking, each m-sized column of $\mathbf{A}$ is a vectorized image or image patch and n is the number of training examples. In some particular models of face recognition and tracking, the number of training examples is even larger. For instance, in [35, 32, 20], the identity matrix $\mathbf{I}_m$ is concatenated with the original dictionary (*i.e.* $n \leftarrow n+m$) to handle partial occlusions at the price of solving a larger LASSO . Also, in sparse subspace clustering [10, 23], the dimensionality of the data points (m) is naturally very large and the task is to cluster the high dimensional data into a set of lower dimensional subspaces. Other applications, such as super resolution [34], tend to have a large number of dictionary elements (n) generated from natural images. Moreover, $\mathbf{A}$ can also have specific structure that can be exploited for faster solutions. For example, in image denoising/restoration [2, 16] and inpainting [30, 26], $\mathbf{A}$ is usually based on a 2D convolutional operator/kernel, thus, resulting in a Block Circulant with Circulant Blocks (BCCB) or Block Topelitz with Topelitz Blocks (BTTB) matrices, where each handles boundary conditions differently. In these cases, $\mathbf{A} \in \mathbb{R}^{n \times n}$ is square and n is the number of pixels in the kernel or filter. Matrix $\mathbf{A}$ gets even "fatter" when multiple kernels or filters are used at the same time, as is the case, in convolutional sparse coding [6, 14, 17].

Solving problem (1) for large-scale $\mathbf{A}$ (especially when m is large) is usually directly coupled with a performance gain for the particular application. This idea has been clearly raised and discussed in face recognition [28, 25]. Despite this gain, these large LASSO problems tend not to be solved because of their significant computational burden. Instead,

Algorithm 1: PL-ADMM for Solving Problem (1)

```
Input  : $\mathbf{b}, \mathbf{A}, \mathbf{c} = \mathbf{0}_n, \mathbf{z} = \mathbf{0}_n, \Psi = \mathbf{0}_m, \lambda, u_1, \gamma > 1.$
Output: $\mathbf{c}$
while not converged do
    c update: solve
    $(2\mathbf{A}^\top \mathbf{A} + u_k \mathbf{I}_n)\mathbf{c}_{k+1} = 2\mathbf{A}^\top \mathbf{b} - \Psi_k + u_k \mathbf{z}_k$
    z update: $\mathbf{z}_{k+1} = \text{soft}(\mathbf{c}_{k+1} + \Psi_k / u_k).$
    Ψ update: $\Psi_{k+1} = \Psi_k + u_k(\mathbf{c}_{k+1} - \mathbf{z}_{k+1})$
    $u_{k+1} = \gamma u_k$
end
```

Algorithm 2: DL-ADMM for Solving Problem (4)

```
Input  : $\mathbf{b}, \mathbf{A}, \mathbf{c} = \mathbf{0}_n, \zeta = \mathbf{0}_n, \Psi = \mathbf{0}_m, \lambda, \rho_1, \gamma > 1$
Output: $\mathbf{c}$
while not converged do
    Ψ update: solve
    $(\rho_k \mathbf{A}\mathbf{A}^\top + \frac{1}{2}\mathbf{I}_m)\Psi_{k+1} = \mathbf{A}(\rho_k \zeta - \mathbf{c}) - \mathbf{b}.$
    ζ update: $\zeta_{k+1} = \text{proj}_{\ell_\infty,\lambda}(\mathbf{A}^\top \Psi_{k+1} + \mathbf{c}_k / \rho_k).$
    c update: $\mathbf{c}_{k+1} = \mathbf{c}_k + \rho_k(\mathbf{A}^\top \Psi_{k+1} - \zeta_{k+1})).$
    $\rho_{k+1} = \gamma \rho_k$
end
```

the columns of $\mathbf{A}$ tend to be downsampled or subsampled to sizes that are manageable by available LASSO solvers. For example, in many sparse trackers, templates of the tracking target that constitute the columns of $\mathbf{A}$ are substantially downsampled (*e.g.* 25-50 times the original patch size) [20]. This strategy has a tradeoff between speed and accuracy.

As for techniques that solve problem (1), many methods abound in the literature and significant advances have been made in this realm. For computational reasons, only first-order LASSO techniques are of interest for large-scale problems. Among these techniques, those that use augmented Lagrangian methods (ALM) for optimization tend to be the most efficient [32, 28, 33]. In this paper, we expand on this type of LASSO solver by lifting problem (1) first and then optimizing its dual form in the Fourier domain using one type of ALM, particularly the Alternating Direction Method of Multipliers (ADMM). This relinquishes the need to solve many large linear systems and affords the use of simpler operations, namely 1D FFT and element-wise vector products. Not only is the computational complexity per ADMM iteration in the lifted domain lower than the state-of-the-art solver, but it is also very trivially parallelizable affording further speedup due to hardware acceleration (*i.e.* simple GPU implementation).

Mathematical Notation. We use boldface lowercase and boldface uppercase letters for vectors and matrices, respectively. $\mathbf{I}_n$ is the identity matrix of size $n \times n$. The operator $\odot$ denotes element-wise products, the FFT of vector $\mathbf{x}$ is denoted as $\hat{\mathbf{x}}$, and $\mathbf{F}$ denotes the normalized discrete Fourier transform (DFT) matrix. Superscripts $*$ and $\mathbf{H}$ indicate a complex conjugation and hermitian operation, respectively. Lastly, the circular convolutional operator is denoted as $\circledast$.

2. Related Work

LASSO solvers are either first- or second-order methods [31, 33]. First-order methods are typically based on local linear approximations with at most linear local error. Examples are not limited to the proximal point [22], parallel coordinate descent [5], and iterative shrinkage thresholding methods (ISTA and FISTA) [1, 7, 22]. Methods of multipliers are also first-order. They include ADMM, which can be applied to problem (1), and we refer to as the primal LASSO (PL-ADMM). Because of LASSO convexity, ADMM can also be applied on the dual problem of (1), which we refer to as the dual LASSO (DL-ADMM). As for second-order methods, they are often computationally expensive including primal-dual interior-point methods [29].

Since first-order methods are very attractive for their computational complexity and decent convergence rates, we briefly discuss some first-order methods to solve problem (1). Problem (1) is an addition of two functions, one smooth and the other non-smooth. A popular class of methods to solve this problem uses iterative soft thresholding (ISTA and FISTA) [1, 7]. Since the smooth part $f(\mathbf{c})$ is gradient Lipschitz continuous, one can bound it with a quadratic function and find a sequence of solutions $\mathbf{c}_k$ that converges to the global optimum by minimizing the upper bound:

$$\begin{aligned} \mathbf{c}_{k+1} = \arg\min_{\mathbf{c}} \ & f(\mathbf{c}_k) + (\mathbf{c} - \mathbf{c}_k)^\top \nabla f(\mathbf{c}_k) \\ & + \frac{1}{2L_f}\|\mathbf{c} - \mathbf{c}_k\|_2^2 + \lambda g(\mathbf{c}), \end{aligned} \tag{2}$$

where $L_f > 0$ is the gradient Lipschitz constant of $f(\mathbf{c})$ (the maximum eigenvalue of $\mathbf{A}^\top \mathbf{A}$). Problem (2) has a closed-form solution (the proximal operator for the ℓ_1 norm):

$$\mathbf{c}_{k+1} = \text{soft}\Big(\mathbf{c}_k - L_f \nabla f(\mathbf{c}_k), L_f \lambda\Big) \tag{3}$$

The iterative soft thresholding method has a sublinear convergence rate. In general, for problems where L_f is taken to be $\frac{1}{\alpha_k}$, it becomes the general proximal point method and the accelarated proximal gradient method (APG). Despite its low computational complexity per iteration, both methods slowly converge in terms of number of iterations [22].

Problem (1) can be solved using coordinate descent, which updates a single entry of the sparse code $\mathbf{c}$ in every iteration. The updates have a closed form [31]. However, it is computationally inefficient overall, as the method requires access to a column of $\mathbf{A}$ in every iteration where element indexing is only partially parallizable on a GPU [9, 31].

Problem (1), can be solved by applying ADMM directly to its primal form, leading to PL-ADMM, or to its dual form, leading to DL-ADMM. As for PL-ADMM, the primal-dual update steps are shown in Algorithm (1). It is important to note that the bottleneck of PL-ADMM is in the $\mathbf{c}$ update because it has to solve a $n \times n$ positive definite (PD) linear system. The soft(.) operator is the standard soft thresholding function. Here, the per iteration complexity is at most $\mathcal{O}(n^3) + \mathcal{O}(mn)$. DL-ADMM optimizes the dual form of problem (1). Note that since LASSO is convex and it satisfies Slater's condition, the duality gap is zero and both the primal and dual forms lead to the same solution. The dual of problem (1) can be easily shown to be:

$$\min_{\Psi} \frac{1}{4}\|\Psi\|_2^2 + \Psi^\top \mathbf{b} \quad \textit{s.t.} \ \|\mathbf{A}^\top \Psi\|_\infty \leqslant \lambda \tag{4}$$

The primal-dual updates of DL-ADMM are shown in Algorithm (2), where $\rho_k > 0$ is a tradeoff coefficient and $\mathbf{c}$ is the primal variable[1]. Similarly, the bottleneck of DL-ADMM is solving a $m \times m$ PD linear system.The per iteration complexity is at most $\mathcal{O}(m^3) + \mathcal{O}(mn)$. Updating ζ is straightforward, as it requires a simple projection onto the ℓ_∞ ball.

It is now clear that the dictionary shape dictates which of the two methods (PL-ADMM vs. DL-ADMM) would be the most computationally efficient. In other words, if $\mathbf{A}$ is tall and skinny (*i.e.* $m > n$), it is more desirable to solve the LASSO using PL-ADMM, while a fat (*i.e.* $m < n$) $\mathbf{A}$ is handled better by DL-ADMM. However, in most sparse representation problems, $\mathbf{A}$ is overcomplete and fat, so we do not focus on solving a LASSO , where m is larger than n. Since most dictionaries are either fat or square (*i.e.* $m \leqslant n$), DL-ADMM is the most efficient option to use for solving a LASSO . In fact, the superiority of DL-ADMM over PL-ADMM (in terms of runtime) has been extensively demonstrated in previous work [32, 28, 33].

Comparison to Other Solvers. In this paper, we focus on solving problem (1) when dictionary $\mathbf{A}$ is square or fat and when *both* m and n are large. Interestingly, when $\mathbf{A}$ is a concatenation of k circulant matrices (*i.e.* $n = km$, as in the case of convolutional sparse coding), the linear system in each DL-ADMM iteration can be easily inverted by computing k 1D FFTs, each of size m. This follows from the fact that circulant matrices can be diagonalized by the DFT matrix $\mathbf{F}$. This modification to DL-ADMM reduces the per iteration time complexity from $\mathcal{O}(m^3) + \mathcal{O}(mn)$ to $\mathcal{O}(nm \log m) + \mathcal{O}(mn)$ and has recently been shown to significantly speedup convolutional sparse coding (CSC) [6, 14] and sparse trackers [35]. Although CSC is a special type of LASSO , we gain inspiration from it to propose our generic LASSO solver, suitably called FFTLasso. We reformulate the LASSO by lifting $\mathbf{c}$ from n to mn dimensions, thus, expanding $\mathbf{A}$ into a block circulant matrix and making LASSO equivalent to a particularly constrained CSC problem. By applying ADMM on the dual of this constrained problem, we observe that the linear system to be solved in each ADMM iteration is block circulant, which can be efficiently solved via 1D FFTs akin to CSC.

Another main distinction between FFTLasso and other LASSO solvers is in how the ADMM dual variable is updated. In all ADMM-based solvers, a conventional gradient ascent step is performed on the dual problem [4]. In FFTLasso and following seminal work on smooth unconstrained optimization [1], we perform a Nesterov *accelerated* gradient ascent step instead, thus, involving the current and past estimates to update the dual variable. This modification substantially reduces (usually by 30%) the number of ADMM iterations needed for convergence.

Contributions. (i) We show that the popular LASSO problem (1) is equivalent to a particularly constrained convolutional sparse coding problem, whose dual can be efficiently solved using ADMM update steps that only require 1D FFTs and element-wise vector operations. As such, the time complexity of our proposed FFTLasso method offers a time complexity of $\mathcal{O}(mn \log m) + \mathcal{O}(mn)$, as opposed to $\mathcal{O}(m^3) + \mathcal{O}(mn)$ in the case of DL-ADMM. Since FFTLasso is very trivially parallelizable, it can easily benefit from hardware (GPU) acceleration. **(ii)** We perform Nesterov accelerated gradient ascent to update the dual variables in FFTLasso. This modification provides a substantial speedup in convergence as compared to conventional ADMM updates. **(iii)** Motivated by a thorough computational analysis and validated by extensive experiments on synthetic and real data, we show that FFTLasso scales much better than the state-of-the-art LASSO solver (DL-ADMM), thus, enabling better performance (*e.g.* face recognition accuracy) in the same amount of time.

3. Proposed Method

In this section, we give a detailed discussion of our FFTLasso solver (see Figure 1). First, we lift the original LASSO and increase the number of variables by appending extra columns in the dictionary $\mathbf{A}$ to generate a larger dictionary $\tilde{\mathbf{A}}$. However, the appended columns have particular structure. We replace each column in $\mathbf{A}$ with all its circular shifts to generate a circulant matrix: $\tilde{\mathbf{A}} = \begin{bmatrix}\mathbf{C}(\mathbf{a}_1) & \cdots & \mathbf{C}(\mathbf{a}_n)\end{bmatrix} \in \mathbb{R}^{m \times mn}$, where $\mathbf{a}_i$ is the i^{th} column of $\mathbf{A}$. The operator $\mathbf{C}(.)$ generates a circulant matrix from a vector. The set of all vectors generated by $\mathbf{C}(\mathbf{x})$ for $\mathbf{x} \in \mathbf{R}^m$ is $\left\{\mathbf{P}^i\mathbf{x} \ \ \forall i = 0, \ldots, m-1\right\}$, where $\mathbf{P}$ is a permutation matrix such that $\mathbf{P}\mathbf{x} = \left[x_m, x_1, \ldots, x_{m-1}\right]^\top$.

Circulant matrices have several nice properties. (1) They are tightly related to circular convolutions: $\mathbf{C}(\mathbf{x})\mathbf{y} = \mathbf{x} \circledast \mathbf{y}$. (2) They can be diagonalized using $\mathbf{F}$, *i.e.* $\mathbf{C}(\mathbf{x}) = \mathbf{F}diag(\hat{\mathbf{x}}^*)\mathbf{F}^{\mathbf{H}}$, where $\hat{\mathbf{x}}^*$ is the conjugate of the 1D FFT

[1] Note that the dual variable of the dual problem is the primal variable.

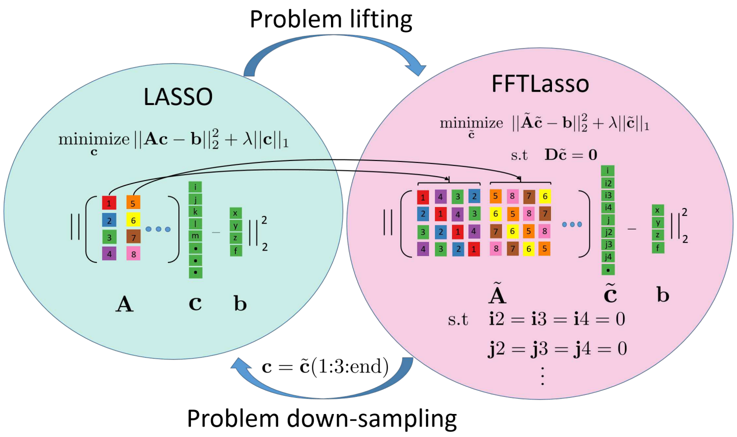

Problem lifting
LASSO
$\underset{\mathbf{c}}{\text{minimize}}\, ||\mathbf{Ac}-\mathbf{b}||_2^2+\lambda||\mathbf{c}||_1$
1 2 3 4 5 6 7 8
i j k l m
x y z f
A
c
b
FFTLasso
$\underset{\tilde{\mathbf{c}}}{\text{minimize}}\ ||\tilde{\mathbf{A}}\tilde{\mathbf{c}}-\mathbf{b}||_2^2+\lambda||\tilde{\mathbf{c}}||_1$
s.t $\mathbf{D}\tilde{\mathbf{c}}=\mathbf{0}$
i i2 i3 i4 j j2 j3 j4
$\tilde{\mathbf{A}}$
$\tilde{\mathbf{c}}$
b
s.t $\mathbf{i2}=\mathbf{i3}=\mathbf{i4}=0$
$\mathbf{j2}=\mathbf{j3}=\mathbf{j4}=0$
$\mathbf{c}=\tilde{\mathbf{c}}(1{:}3{:}\text{end})$
Problem down-sampling

Figure 1. Relationship between FFTLasso and traditional LASSO .

of $\mathbf{x}$. Therefore, it is easy to show that problem (1) is equivalent to the following:

$$\min_{\tilde{\mathbf{c}}} \|\tilde{\mathbf{A}}\tilde{\mathbf{c}}-\mathbf{b}\|_2^2+\lambda\|\tilde{\mathbf{c}}\|_1 \quad \text{s.t} \quad \mathbf{D}\tilde{\mathbf{c}}=\mathbf{0}, \tag{5}$$

where $\mathbf{D} \in \mathbb{R}^{n(m-1)\times mn}$ is a selection matrix that chooses the values of $\tilde{\mathbf{c}}$ corresponding to the appended columns in $\tilde{\mathbf{A}}$ and sets them to 0. It is also important to note that problem (5) can be rewritten as follows:

$$\min_{\tilde{\mathbf{c}}} \left\|\sum_i^n \mathbf{a}_i \circledast \tilde{\mathbf{c}}_i-\mathbf{b}\right\|_2^2+\lambda\|\tilde{\mathbf{c}}\|_1 \ \text{s.t} \ \mathbf{D}\tilde{\mathbf{c}}=\mathbf{0}, \tag{6}$$

where $\tilde{\mathbf{c}}^\top = \left[\tilde{\mathbf{c}}_1^\top,\ldots,\tilde{\mathbf{c}}_n^\top\right]^\top$. Note that our LASSO reformulation can be seen as a constrained convolutional sparse coding (CSC), where the sparse codes are constrained to have the following structure: $\tilde{\mathbf{c}}_i^\top = \left[c_i,\ \mathbf{0}_{m-1}^\top\right]^\top \forall i$ and where c_i is the i^{th} element of the LASSO solution of problem (1). Therefore, it is easy to realize that $\mathbf{a}_i \circledast \tilde{\mathbf{c}}_i = \mathbf{a}_i c_i$ and $\sum_i^n \mathbf{a}_i \circledast \tilde{\mathbf{c}}_i = \sum_i^n \mathbf{a}_i c_i = \mathbf{Ac}$, demonstrating that LASSO is indeed a CSC problem with special constraints.

Solving Problem (5). Now, we focus on developing an efficient solver for our LASSO reformulation in problem (5). At first glance, this optimization is much larger than the original LASSO and it has many linear equality constraints, two characteristics that one usually avoids. However, we will show that the dual of this problem can be solved efficiently using simple ADMM steps (unlike DL-ADMM), which do not require operators that are more computationally expensive than 1D FFTs, and that the linear selection constraints can be trivially handled. It is easy to show that the dual of problem (5) is as follows:

$$\min_{\Psi,\theta} \frac{1}{4}\|\Psi\|_2^2+\Psi^{\mathbf{H}}\mathbf{b} \ \text{s.t} \ \|\tilde{\mathbf{A}}^{\mathbf{H}}\Psi+\mathbf{D}^{\mathbf{H}}\theta\|_\infty \leqslant \lambda, \tag{7}$$

where Ψ is the dual variable. To solve problem (7) using ADMM, we first add an auxiliary variable: $\zeta = \tilde{\mathbf{A}}^{\mathbf{H}}\Psi + \mathbf{D}^{\mathbf{H}}\theta$, to separate the ℓ_∞ constraint from Ψ. Therefore, we define the augmented Lagrangian function $\mathcal{L}_\rho$ as follows:

```
Algorithm 3: FFTLasso for Solving Problem (1)
Input  : $\mathbf{b}, \mathbf{A}, \tilde{\mathbf{c}}_1 = \tilde{\mathbf{y}}_1 = \tilde{\mathbf{r}}_1 = \mathbf{e}_1 = \mathbf{t}_1 = \zeta_1 =$
         $\mathbf{D}^{\mathbf{H}}\theta_1 = \mathbf{0}_{mn}, \Psi = \mathbf{0}_m, \lambda, \rho_1, \gamma > 1, q.$
Output: $\mathbf{c}$
while not converged do
    compute: $\mathbf{e}_{k+1} = \rho_k\zeta_k - \rho_k\mathbf{D}^{\mathbf{H}}\theta_k - \tilde{\mathbf{c}}_k$
    $\hat{\Psi}^*$ update: $\hat{\Psi}^*_{k+1} = \frac{\sum_i^N \hat{\mathbf{a}}_i^* \odot \hat{\mathbf{e}}^*_{ik+1} - \hat{\mathbf{b}}^*}{\rho_k \sum_i^N \hat{\mathbf{a}}_i \odot \hat{\mathbf{a}}_i^* + \frac{1}{2}\mathbf{1}_m}$
    compute: $\tilde{\mathbf{A}}^{\mathbf{H}}\Psi_{k+1}$, see Eq (12)
    $\mathbf{D}^{\mathbf{H}}\theta$ update:
    $(\mathbf{D}^{\mathbf{H}}\theta_{k+1}) = (\zeta_k - \frac{1}{\rho_k}\tilde{\mathbf{c}}_k - \tilde{\mathbf{A}}^{\mathbf{H}}\Psi_{k+1})$
    $(\mathbf{D}^{\mathbf{H}}\theta_{k+1})_{\text{1:m:end}} = \mathbf{0}_n$
    compute: $\mathbf{t}_{k+1} = \tilde{\mathbf{A}}^{\mathbf{H}}\Psi_{k+1} + \mathbf{D}^{\mathbf{H}}\theta_{k+1} + \tilde{\mathbf{c}}_k/\rho_k$
    $\zeta$ update: $\zeta_{k+1} = \text{sign}(\mathbf{t}_{k+1}) \odot \min(|\mathbf{t}_{k+1}|, \lambda)$
    $\tilde{\mathbf{c}}$ update:
    $\tilde{\mathbf{c}}_{k+1} = \tilde{\mathbf{y}}_k + \rho_k(\tilde{\mathbf{A}}^{\mathbf{H}}\Psi_{k+1} + \mathbf{D}^{\mathbf{H}}\theta_{k+1} - \zeta_{k+1})$
    compute: $\tilde{\mathbf{y}}_{k+1} = (1+q)\tilde{\mathbf{c}}_{k+1} - q\tilde{\mathbf{c}}_k$
    $\rho_{k+1} = \gamma\rho_k$
end
$\mathbf{c} \leftarrow \tilde{\mathbf{c}}(1{:}m{:}\text{end})$
```

$$\begin{aligned}\mathcal{L}_\rho(\Psi,\theta,\zeta,\tilde{\mathbf{c}}) &:= \frac{1}{4}\|\Psi\|_2^2+\Psi^{\mathbf{H}}\mathbf{b}+\mathbb{I}_{\{\|\zeta\|_\infty\leqslant\lambda\}}+\\ &\tilde{\mathbf{c}}^{\mathbf{H}}(\tilde{\mathbf{A}}^{\mathbf{H}}\Psi+\mathbf{D}^{\mathbf{H}}\theta-\zeta)+\frac{\rho}{2}\|\tilde{\mathbf{A}}^{\mathbf{H}}\Psi+\mathbf{D}^{\mathbf{H}}\theta-\zeta\|_2^2\end{aligned} \tag{8}$$

where $\tilde{\mathbf{c}}$ is also the vector of Lagrange multipliers corresponding to the ζ constraint. Because of the zero duality gap, one can show that the primal solution $\tilde{\mathbf{c}}$ of problem (5) is indeed the optimal dual variable of this problem's dual formulation. $\mathbb{I}$ is the indicator function that penalizes infeasible ζ, and $\rho \geqslant 0$ is the ADMM penalty parameter.

In what follows, we elaborate on each ADMM update, which first minimizes $\mathcal{L}_\rho$ w.r.t. each primal variable (Ψ, θ, and ζ) separately and then updates the dual variable $\tilde{\mathbf{c}}$ via an accelerated Nesterov dual ascent step. All these steps are summarized in Algorithm (3).

Update $\hat{\Psi}^*$: We need to solve the following PD system:

$$(\rho\tilde{\mathbf{A}}\tilde{\mathbf{A}}^{\mathbf{H}}+\frac{1}{2}\mathbf{I}_m)\Psi = \tilde{\mathbf{A}}(\rho\zeta-\rho\mathbf{D}^{\mathbf{H}}\theta-\tilde{\mathbf{c}})-\mathbf{b} \tag{9}$$

We can compute the right hand side of this system efficiently. Setting $\mathbf{e} = \rho\zeta-\rho\mathbf{D}^{\mathbf{H}}\theta-\tilde{\mathbf{c}} = \left[\mathbf{e}_1^{\mathbf{H}},\ldots,\mathbf{e}_n^{\mathbf{H}}\right]^{\mathbf{H}}$, we have $\tilde{\mathbf{A}}\mathbf{e} = \sum_i^n \mathbf{C}(\mathbf{a}_i)\mathbf{e}_i = \mathbf{F}\sum_i^n \hat{\mathbf{a}}_i^* \odot \hat{\mathbf{e}}_i^*$. Thus, it is easy to see that the update for $\hat{\Psi}^*$ (conjugate of FFT of Ψ) is as:

$$\hat{\Psi}^* = \frac{\sum_i^N \hat{\mathbf{a}}_i^* \odot \hat{\mathbf{e}}_i^* - \hat{\mathbf{b}}^*}{\rho\sum_i^N \hat{\mathbf{a}}_i \odot \hat{\mathbf{a}}_i^* + \frac{1}{2}\mathbf{1}_m}, \tag{10}$$

where the vector division is done element-wise in the Fourier domain. For this update, we only need to compute the

m-sized FFT of each of the n parts of $\mathbf{e}$, since the FFT of $\mathbf{b}$ and each $\mathbf{a}_i$ is done once before ADMM commences.

Update $\mathbf{D^H}\theta$: We do not need to update θ, since it never appears alone in any ADMM step. Instead, we update $\mathbf{D^H}\theta$ by solving the following linear system:

$$\mathbf{D}(\mathbf{D^H}\theta) = \mathbf{D}(\zeta - \tilde{\mathbf{c}}/\rho - \tilde{\mathbf{A}}^{\mathbf{H}}\Psi) \quad (11)$$

To compute $\mathbf{D^H}\theta$, we compute $\tilde{\mathbf{A}}^{\mathbf{H}}\Psi$ efficiently in the Fourier domain. The diagonalization trick can be applied again for each block of $\tilde{\mathbf{A}}$ as follows:

$$\tilde{\mathbf{A}}^{\mathbf{H}}\Psi = \begin{bmatrix} \mathbf{C}(\mathbf{a}_1)^{\mathbf{H}} \\ \vdots \\ \mathbf{C}(\mathbf{a}_n)^{\mathbf{H}} \end{bmatrix} \Psi = \begin{bmatrix} \mathbf{F}(\hat{\mathbf{a}}_1 \odot \hat{\Psi}^*) \\ \vdots \\ \mathbf{F}(\hat{\mathbf{a}}_n \odot \hat{\Psi}^*) \end{bmatrix} \quad (12)$$

Since $\mathbf{D}$ is merely a selection matrix for the appended columns in $\tilde{\mathbf{A}}$, we simply set the corresponding $n(m-1)$ elements of $\mathbf{D^H}\theta$ to those of the right hand side. The rest of the elements in $\mathbf{D^H}\theta$ remain intact.

Update ζ : This is done by solving problem (13).

$$\min_{\zeta} \ \|\zeta - (\tilde{\mathbf{A}}^{\mathbf{H}}\Psi + \mathbf{D^H}\theta + \tilde{\mathbf{c}}/\rho)\|_2^2 \ \text{ s.t. } \|\zeta\|_\infty \leqslant \lambda \quad (13)$$

Problem (13) is a Eucledian projection of $\mathbf{t} = \tilde{\mathbf{A}}^{\mathbf{H}}\Psi + \mathbf{D^H}\theta + \tilde{\mathbf{c}}/\rho$ onto the ℓ_∞ ball. It can be computed using element-wise operators (min and sign) as follows:

$$\zeta = \text{sign}(\mathbf{t}) \odot \min(|\mathbf{t}|, \lambda) \quad (14)$$

Update $\tilde{\mathbf{c}}$: Conventionally, the dual variable $\tilde{\mathbf{c}}$ should be updated by a dual ascent step as follows:

$$\tilde{\mathbf{c}} \leftarrow \tilde{\mathbf{c}} + \rho(\tilde{\mathbf{A}}^{\mathbf{H}}\Psi + \mathbf{D^H}\theta - \zeta) \quad (15)$$

However, we propose to speed up the convergence of our ADMM method by replacing the typical dual ascent step above with Nesterov's accelerated ascent step as follows:

$$\begin{aligned} \tilde{\mathbf{c}}_{k+1} &\leftarrow \tilde{\mathbf{y}}_k + \rho(\tilde{\mathbf{A}}^{\mathbf{H}}\Psi + \mathbf{D^H}\theta - \zeta) \\ \tilde{\mathbf{y}}_{k+1} &\leftarrow (1+q)\tilde{\mathbf{c}}_{k+1} - q\tilde{\mathbf{c}}_k \end{aligned} \quad (16)$$

where k is the iteration number and $q = \frac{\sqrt{Q}-1}{\sqrt{Q}+1}$ is set according to [21]. Here, Q represents the condition number; however, we set it to a constant throughout all the experiments for convenience. Note that when $Q = 1$, the update in (16) degenerates to the standard dual ascent update in (15). In fact, the Nesterov accelerated gradient ascent method has been shown to afford a significant speedup in convergence, when it is compared to regular gradient ascent [13]. We exploit this property to reduce the number of ADMM iterations needed for convergence, which is guaranteed regardless of the initialization due to the convexity of the LASSO problem.

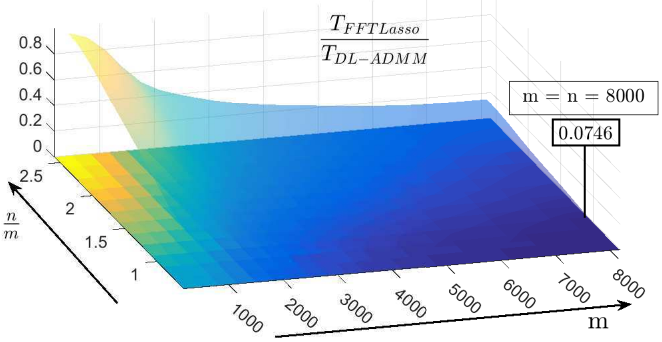


Figure 2. Ratio of iteration runtime between FFTLasso and DL-ADMM. Only when the dictionary is small and has many more columns than rows is DL-ADMM competitive with FFTLasso.

The details of FFTLasso are summarized in Algorithm (3). For a complete mathematical treatment, and for an efficient implementation with matrix notation we refer the reader to the **supplementary material**.

Computational Analysis. By studying the DL-ADMM method, we know that its per-iteration time complexity is $\mathcal{O}(m^3) + \mathcal{O}(mn)$, while it is $\mathcal{O}(mn \log m) + \mathcal{O}(mn)$ for our FFTLasso. The most expensive computational operation needed in FFTLasso is n 1D FFTs of size m each. The lifting procedure at the beginning allows our method to avoid using expensive linear solvers in its updates and is primarily what differentiates us from DL-ADMM in runtime. Since we set ρ to scale proportionally with mn, we empirically observe that the number of ADMM iterations needed for FFTLasso to converge, although much higher than DL-ADMM, effectively does not grow with mn. We can use this result to comment on the scale of problems where FFTLasso is expected to be more efficient than DL-ADMM. In fact, dictionaries with $m^2 \gg n \log m$ are best suited for FFTLasso, where square matrices naturally satisfy inequality. This means that FFTLasso favors a dictionary $\mathbf{A}$, whose number of columns is not tremendously more than the number of its rows, while DL-ADMM is tailored for much fatter dictionaries. This is evident from Figure (2), where the computational complexity per iteration of FFTLasso consistently improves (reaching a 13× speedup) as compared to DL-ADMM, especially when both (m, n) are large.

4. Experiments

In this section, we conduct extensive experiments to motivate and evaluate our proposed formulation both on synthetic data followed by experiments on a popular computer vision task (face recognition). The section is organized as follows: (1) parameters and implementation details; (2) speed comparison on different dictionaries sizes and (3) face recognition results.

Dataset preparation. We conduct experiments on both a synthetic and real face dataset. In the synthetic experiments, we start by generating square dictionaries $\mathbf{A} \in \mathbb{R}^{n \times n}$ and regressors $\mathbf{b} \in \mathbb{R}^n$ for $n \in \{10^3, 1500, 2000, ..., 10^4\} \cup \{2^9, 2^{10}, ..., 2^{13}\}$. All dictionaries and regressors are i.i.d. Gaussian matrices and vectors, respectively. Following convention [31, 33], the columns of $\mathbf{A}$ and $\mathbf{b}$ have unit norm.

As for the face recognition task, large datasets with prealigned and cropped faces are not readily publicly available. Therefore, we conduct our experiments on part of the extended Yale B dataset [12], which contains a total of 16,128 face images from 28 human subjects. The images of each subject are taken under 9 different poses and 64 different illumination conditions. Unfortunately, since the faces are not cropped nor aligned, we run an off-the-shelf face detector to crop a total of 10,140 images distributed uniformly across the 28 subjects. The average face size is 245 × 245 pixels. We conduct our experiments at two dimensions for a non-square dictionary $\mathbf{A}$. In both experiments, we use $n = 10^4$ training faces, whose face patches are downsampled and vectorized to $m \in \{2^{12}, 2^{13}\}$ pixels. Note that the largest face size used in related work [33, 31] is 40 × 30 or 1,200 pixels, which is 4 times smaller than the smallest m in our experiments. Following convention, five random test faces are chosen from each class, resulting in 140 test samples (*i.e.* 140 values of $\mathbf{b}$). For each test sample $\mathbf{b}$, a LASSO is solved and its sparse code $\mathbf{c}$ is computed. Following convention, we assign $\mathbf{b}$ to subject i^* that leads to the smallest residual: $i^* = \arg\min_i \|\mathbf{A}\mathbf{c}_i - \mathbf{b}\|_2^2$, where $\mathbf{c}_i$ is the sparse code corresponding to subject i only.

Parameters and implementation details. For a fair comparison, we choose the ADMM parameters with the fastest convergence for both methods at $m = 2^{13}$ on the synthetic and face dataset. Then, these parameters are scaled (in the same way) according to the dimension of the problem. For DL-ADMM, when $m = n = 2^{13}$, we set $\rho_1 = \sqrt{10}$ and increase it as $\rho_{k+1} = \min\{\gamma\rho_k, \rho_{\max}\}$, where $\gamma = \sqrt{1.001}$ and $\rho_{\max} = \sqrt{2000}$. As for FFTLasso, the corresponding parameters for the same dimension m are $\rho_1 = 70$, $\gamma = 1.0005$ and $\rho_{\max} = 2000$. When the m value changes, all parameters for both methods are kept the same, except for ρ_1, which is proportionally scaled with m.

The synthetic experiments aim to show the convergence rate of each method, so a high accuracy solution $\mathbf{c}_o$ is obtained using a MATLAB mex-function [19, 18] of the LARS algorithm [8]. Following the evaluation paradigm of [33, 31], we terminate DL-ADMM and FFTLasso, once they reach a low relative argument error w.r.t. $\mathbf{c}_o$: $\|\mathbf{c} - \mathbf{c}_o\| \leqslant \tau\|\mathbf{c}_o\|$. In our experiments, we set τ=1%. We set λ for each dictionary, so as to achieve $\approx 60\%$ sparsity for the synthetic experiments and $\approx 85\%$ for the face dataset. In the face recognition experiments, we let DL-ADMM and FFTLasso run for the same amount of time (10 minutes), where the intermediary argument relative error, the recognition accuracy, and the objective error are reported.

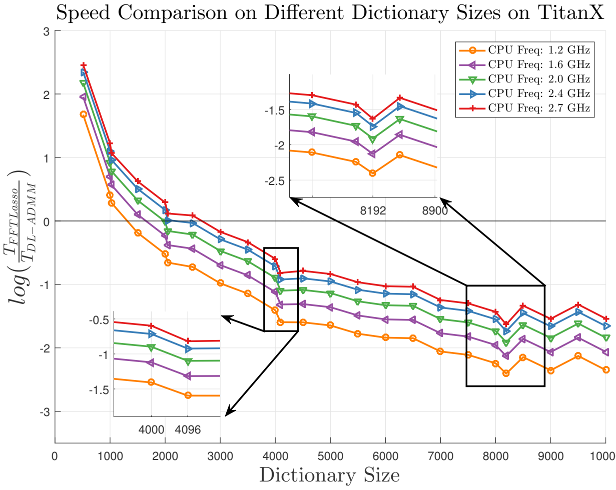


Figure 3. Runtime comparison between FFTLasso-GPU (on TitanX) and DL-ADMM (multi-core) at different CPU frequencies.

4.1. Comparison on Synthetic Data

In this section, we conduct three experiments, each of which studies the LASSO problem from a different perspective. In the first experiment, we compare the convergence rate between a MATLAB CPU implementation of DL-ADMM as given in Algorithm (2), against a simple GPU implementation of FFTLasso, where the FFTs required in Algorithm (3) are done in parallel. In the second experiment, we demonstrate the impact of using Nesterov accelerated gradient ascent for both DL-ADMM and FFTLasso. In this case, we use a simple GPU implementation of DL-ADMM (similar to FFTLasso). FFTLasso's memory efficiency is established in the last experiment.

1. Convergence Comparison. To the best of our knowledge, all mainstream solvers for DL-ADMM are implemented using a CPU. Also, implementations of many linear algebra routines (*e.g.* matrix inversion and linear system solving) are highly optimized on the CPU in MATLAB, allowing it to benefit from multi-core high-frequency CPUs that are available nowadays. Due to these two reasons and only in this experiment, we compare a multi-core CPU implementation of DL-ADMM at different frequencies against the GPU version of FFTLasso, denoted FFTLasso-GPU. Using a GPU exploits the trivial parallelism in FFTLasso, namely the parallel computation of the 1D FFTs, where we use a TitanX (1.0GHz/core) for this experiment[2]. Here, DL-ADMM runs on 8 cores with varying speeds {1.2, 1.6, 2.0, 2.4, 2.7} GHz/core. All experiments are in single precision.

Figure 3 shows a speed comparison (in log scale) between DL-ADMM (on multiple cores) with varying CPU frequency against FFTLasso-GPU on TitanX. We plot the log ratio of the two methods as $\log\left(T_{\text{FFTLasso}}/T_{\text{DL-ADMM}}\right)$.

[2] Results on the TitanX Pascal GPU (1.4GHz/core) are left to the **supplementary material**.

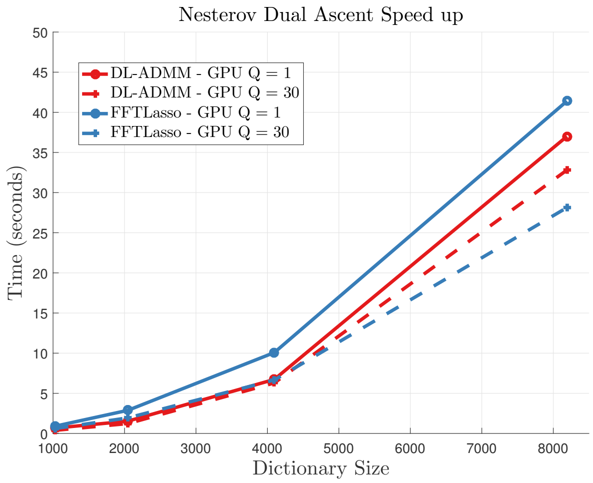


Figure 4. Comparing DL-ADMM against FFTLasso on the same GPU with and without Nesterov acceleration. Q=1 indicates regular dual ascent and Q=30 indicates incorporated acceleration.

FFTLasso-GPU benefits from GPU hardware acceleration much more than DL-ADMM does from multi-core CPU acceleration. It is also clear that FFTLasso-GPU significantly outperforms DL-ADMM for all dimensions $m = n > 3000$, even though MATLAB has a very efficient linear solver for dense matrices. This is mainly because the linear solver (see Algorithm 2) scales poorly (cubically) with m, as compared to FFTLasso that only requires independent 1D FFT operations per ADMM iteration. It is also important to highlight the non-monotonic decreasing nature of the plot. We observe that the runtime gain of FFTLasso over DL-ADMM can decrease with m (visualized as jumps in the plot), especially at values of m that are *not* powers of 2. This is due to the fact that the FFT routine we use has radix 2, so it is most efficient when $m = 2^v$ for an integer value v. In fact, applying FFTLasso for m slightly larger than 2^v (*e.g.* $2^v + \epsilon$) would require the computation of FFTs of size 2^{v+1} (by zeroing padding), thus, incurring unnecessary extra computation and decreasing the speed of FFTLasso. Of course, FFT routines with higher radix can be used to alleviate this. Since FFTLasso is best suited for radix 2, we will only consider these dimensions in the rest of the paper.

2. Nesterov Speed Up. Here, we demonstrate the effect of using a Nesterov accelerated gradient ascent step instead of the conventional dual ascent step in both FFTLasso and even DL-ADMM. From this experiment onwards, we will be comparing FFTLasso-GPU to DL-ADMM-GPU, which is a GPU implementation of DL-ADMM similar to that of FFTLasso-GPU. The results of this runtime comparison are shown in Figure 4. Setting $Q = 1$ is equivalent to regular dual ascent (no acceleration). However, when $Q = 30$, both DL-ADMM-GPU and FFTLasso-GPU experience a significant boost in speed, since a fewer number of ADMM iterations are needed for convergence. At $m = n = 2^{13}$, our method requires 35% less iterations due to the Nesterov acceleration, while this reduction is 13% for DL-ADMM. When comparing our proposed solver (FFTLasso-GPU with Nesterov) to conventional DL-ADMM-GPU, we observe that the runtime gap between the two methods increases with m, reaching 37% at $m = 2^{13}$.

Table 1. Memory efficiency comparison between DL-ADMM and FFTLasso on a TitanX GPU with 12GB memory. Dimensions marked with ✓(*) are reached using simple dictionary splitting that is only possible for FFTLasso. Refer to the text for details.

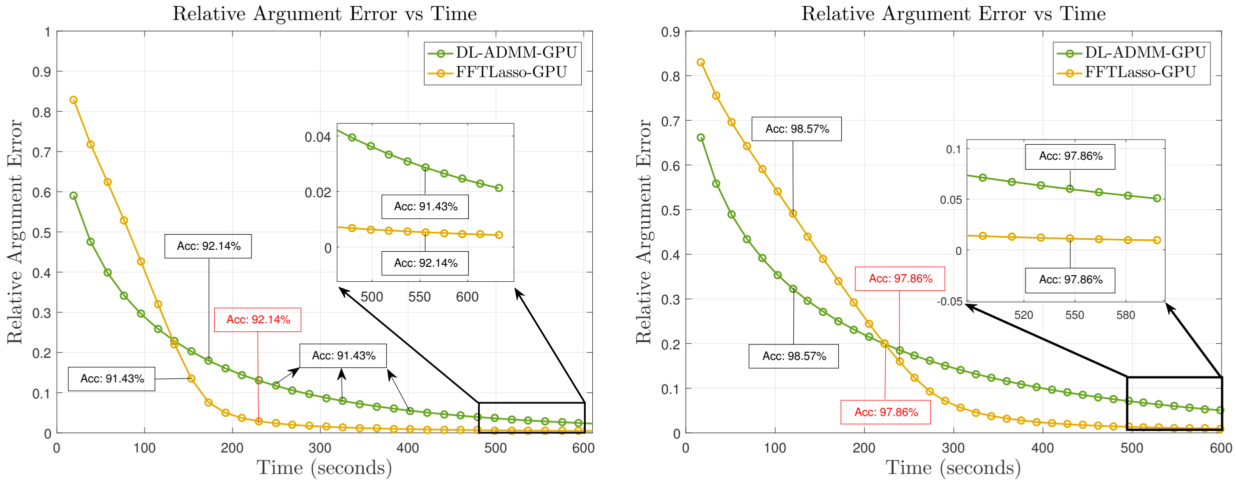
Dictionary Size	2^{12}	6500	2^{13}	8500	9000	9500
DL-ADMM	✓	✓	✗	✗	✗	✗
FFTLasso	✓	✓	✓	✓	✓(*)	✓(*)

3. Memory Efficiency. To compare the memory efficiency of both DL-ADMM and FFTLasso, we run both methods on different sized dictionaries in double precision. As shown in Table (1), DL-ADMM cannot handle dictionaries where $m = n > 6500$ due to the overhead in solving the linear system in Algorithm (2). However, the 1D FFT operation has less memory overhead, so FFTLasso handles much bigger dictionaries reaching $m = n = 8500$, where it breaks. Interestingly, FFTLasso's memory efficiency can be trivially extended to handle dictionaries of larger size (reaching $m = n = 9500$) by horizontally splitting the dictionary $\mathbf{A}$ into two pieces $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{R}^{\frac{m}{2} \times n}$ and applying $\frac{m}{2}$ FFTs on the columns of $\mathbf{A}_1$ and $\mathbf{A}_2$ independently, thus, minimizing memory overhead. The two FFTs of $\mathbf{A}_1$ and $\mathbf{A}_2$ can be merged later only using replication and multiplication with complex exponentials. This simple extension is well-known and trivial to implement; however, it cannot be trivially carried over to solving large linear systems (DL-ADMM). We refer the reader to the **supplementary material** for details.

4.2. Face Recognition

Two main experiments are conducted for the face recognition task. The dictionaries $\mathbf{A}$ and the test samples $\mathbf{b}$ are based on the YaleB face dataset [12], where the dictionary size is $(m = 2^{12}, n = 10^4)$ in one experiment and $(m = 2^{13}, n = 10^4)$ in another. Note that the face images are downsampled more in the first experiment than the other. For fair runtime comparison between DL-ADMM (GPU implementation) and FFTLasso (GPU implementation), we run both for the same amount of time (600 seconds) for every test sample. Intermediary solutions are recorded during the optimization at uniform time intervals.

In Figure 5, we plot both the average relative argument error among all 140 test samples at each time interval, as well as, the recognition accuracy on the test set at some of these intervals for both dictionaries. We denote the recognition accuracy obtained when $\mathbf{c}_o$ (*i.e.* the solution both FFTLasso and DL-ADMM should converge to) is used as the *target accuracy*. For the smaller dictionary size $(m = 2^{12}, n = 10^4)$, Figure 5(a) compare the convergence behav-



(a) $m = 2^{12}, n = 10^4$.

(b) $m = 2^{13}, n = 10^4$.

Figure 5. Comparison between DL-ADMM-GPU and FFTLasso-GPU on the YaleB face dataset with two different dictionaries. Figures (a,b) demonstrate the convergence rate of the relative argument error with the corresponding accuracy for the datasets of sizes ($m = 2^{12}, m = 2^{13}$) respectively. Instances of accuracies indicated in red are the first instances where a solver converged to the $\mathbf{c}_o$ accuracy.

ior of both methods w.r.t. relative argument error w.r.t. $\mathbf{c}_o$. In this case, the target accuracy is 92.14%. It is clear that FFTLasso converges much faster than DL-ADMM in argument error and reaches 92.14% in ≈ 230 seconds, while it takes more than 600 seconds for DL-ADMM to converge to the same accuracy. Similarly, we consider the larger dictionary ($m = 2^{13}, n = 10^4$) in the second experiment and Figure 5(b) summarizes the results. Here, the target accuracy is 97.86%. FFTLasso and DL-ADMM both reach the target accuracy in ≈ 240 seconds; however, FFTLasso has much faster convergence in argument error as compared to DL-ADMM. It is important to note that the relative speedup of FFTLasso w.r.t. DL-ADMM increases from what it is in the first experiment. This verifies what our comparative computational analysis concluded earlier, *i.e.* FFTLasso will be faster than DL-ADMM when the dictionary $\mathbf{A}$ grows larger so long both dimensions grow in a reasonably similar way.

Also, the time required for either DL-ADMM or FFTLasso to converge to high precision relative argument error is much higher than the case for the synthetic experiment in Figure 4. This is due to the high coherence of the columns of the dictionary $\mathbf{A}$ as opposed to the random dictionary. An important result from these experiments is that the accuracy obtained for the larger dimension dictionary (*i.e.* when face images are subsampled less) is 5% higher than the one obtained for the smaller dictionary (*i.e.* when face images are subsampled more). A similar conclusion is reached in previous work on smaller subsets of YaleB [32, 28, 33]. We are aware that the faces in our dataset are not perfectly cropped and aligned (due to errors in the automated face detector), reaching an impressive accuracy of 97.86% is now possible by solving a larger LASSO . Thus, having an efficient-scalable LASSO solver maps to higher accuracy.

5. Discussion

Now, we summarize the types of dictionaries that FFTLasso is best suited for. In general, FFTLasso is the preferred LASSO solver for large-scale square and close-to-square ($m^2 \gg n \log m$) dictionaries. If the dictionary $\mathbf{A}$ fits in GPU memory, FFTLasso is best suited for the case when $m = 2^v$. If $\mathbf{A}$ is too big to fit in memory and/or no available linear solver can handle its size, the dictionary can be split into smaller pieces and FFTLasso can be trivially distributed across multiple GPUs (or even CPUs), due to the parallelizable nature of the updates in Algorithm (3).

6. Conclusions

In this paper, we propose a new equivalent formulation to the Lasso problem to handle large scale dictionaries. The new formulation is a special case of constrained convolutional sparse coding. All the updates of the subproblems are element-wise operations in the Fourier domain where no linear solvers nor a matrix vector multiplication is needed leading to a computation effective updates. The proposed FFTLasso can be trivially benefit from a GPU hardware acceleration. Also, Nesterov's accelerated gradient is applied to standard ADMM and demonstrated the potential for faster convergence. Experiments on synthetic and real data have been conducted to verify the conclusions.

Acknowledgments. This work was supported by the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research. We thank Congli Wang for his help in the CUDA implementation.

References

[1] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[2] J. Bect, L. Blanc-Féraud, G. Aubert, and A. Chambolle. A l 1-unified variational framework for image restoration. In *European Conference on Computer Vision*, pages 1–13. Springer, 2004.

[3] A. Bibi, T. Zhang, and B. Ghanem. 3d part-based sparsea tracker with automatic synchronization and registration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1439–1448, 2016.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[5] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*, 2011.

[6] H. Bristow, A. Eriksson, and S. Lucey. Fast convolutional sparse coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 391–398, 2013.

[7] D. L. Donoho and Y. Tsaig. Fast solution of-norm minimization problems when the solution may be sparse. *IEEE Transactions on Information Theory*, 54(11):4789–4812, 2008.

[8] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, et al. Least angle regression. *The Annals of statistics*, 2004.

[9] M. Elad, B. Matalon, and M. Zibulevsky. Coordinate and subspace optimization methods for linear least squares with non-quadratic regularization. *Applied and Computational Harmonic Analysis*, 23(3):346–367, 2007.

[10] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009.

[11] E. Elhamifar and R. Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE transactions on pattern analysis and machine intelligence*, 2013.

[12] A. Georghiades, P. Belhumeur, and D. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 23(6):643–660, 2001.

[13] T. Goldstein, B. O'Donoghue, S. Setzer, and R. Baraniuk. Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, 7(3):1588–1623, 2014.

[14] F. Heide, W. Heidrich, and G. Wetzstein. Fast and flexible convolutional sparse coding. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5135–5143. IEEE, 2015.

[15] Z. Hong, X. Mei, D. Prokhorov, and D. Tao. Tracking via robust multi-task multi-view joint sparse representation. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 649–656. IEEE, 2013.

[16] Y. Huang, M. K. Ng, and Y.-W. Wen. A fast total variation minimization method for image restoration. *Multiscale Modeling & Simulation*, 7(2):774–795, 2008.

[17] B. Kong and C. C. Fowlkes. Fast convolutional sparse coding. Technical report, Department of Computer Science, University of California, Irvine, 2014.

[18] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696. ACM, 2009.

[19] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(Jan):19–60, 2010.

[20] X. Mei and H. Ling. Robust visual tracking using &# x2113; 1 minimization. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009.

[21] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[22] N. Parikh, S. P. Boyd, et al. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

[23] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.

[24] A. Wagner, J. Wright, A. Ganesh, Z. Zhou, and Y. Ma. Towards a practical face recognition system: Robust registration and illumination by sparse representation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 597–604. IEEE, 2009.

[25] A. Wagner, J. Wright, A. Ganesh, Z. Zhou, H. Mobahi, and Y. Ma. Toward a practical face recognition system: Robust alignment and illumination by sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 372–386, 2012.

[26] Y. Wang, J. Yang, W. Yin, and Y. Zhang. A new alternating minimization algorithm for total variation image reconstruction. *SIAM Journal on Imaging Sciences*, 1(3):248–272, 2008.

[27] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T. S. Huang, and S. Yan. Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE*, 98(6):1031–1044, 2010.

[28] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, pages 210–227, 2009.

[29] S. J. Wright. *Primal-dual interior-point methods*. Siam, 1997.

[30] Z. Xu and J. Sun. Image inpainting by patch propagation using patch sparsity. *IEEE transactions on image processing*, pages 1153–1165, 2010.

[31] A. Y. Yang, A. Genesh, Z. Zhou, S. S. Sastry, and Y. Ma. A review of fast l (1)-minimization algorithms for robust face recognition. Technical report, DTIC Document, 2010.

[32] A. Y. Yang, S. S. Sastry, A. Ganesh, and Y. Ma. Fast l 1-minimization algorithms and an application in robust face recognition: A review. In *2010 IEEE International Conference on Image Processing*. IEEE, 2010.

[33] A. Y. Yang, Z. Zhou, A. G. Balasubramanian, S. S. Sastry, and Y. Ma. Fast-minimization algorithms for robust face

recognition. *IEEE Transactions on Image Processing*, pages 3234–3246, 2013.

[34] J. Yang, J. Wright, T. Huang, and Y. Ma. Image super-resolution as sparse representation of raw image patches. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[35] T. Zhang, A. Bibi, and B. Ghanem. In defense of sparse tracking: Circulant sparse tracker. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3880–3888, 2016.

[36] T. Zhang, B. Ghanem, S. Liu, and N. Ahuja. Robust visual tracking via structured multi-task sparse learning. *International journal of computer vision*, pages 367–383, 2013.

[37] T. Zhang, K. Jia, C. Xu, Y. Ma, and N. Ahuja. Partial occlusion handling for visual tracking via robust part matching. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1258–1265. IEEE, 2014.

[38] T. Zhang, S. Liu, C. Xu, S. Yan, B. Ghanem, N. Ahuja, and M.-H. Yang. Structural sparse tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 150–158, 2015.