Adaptive Accelerations for Learning-based Sparse Coding

Zheng Zhou
Fuzhou University
College of Physics and Information Engineering, China
N181120083@fzu.edu.cn

Tomoya Sakai
Nagasaki University
Graduate School of Engineering, Japan
tsakai@cis.nagasaki-u.ac.jp

Abstract

Sparse coding is an important paradigm to represent visual information, however, many representation-based recognition tasks are still limited by sparse solvers due to their effective computational cost. In this paper, we study data-driven learning for iterative algorithms to accelerate the approximation of sparse solutions. We propose a strategy for sparse dictionary learning combined with automatic tuning of step sizes in the iterative soft thresholding algorithm (ISTA) via its unfolding into a computational graph. In order to make it practical in large-scale applications, we suggest learning the step sizes instead of tuning them. Furthermore, following the assumption of the existence of a data-dependent "optimum" in the acceleration technique implemented in the fast iterative soft thresholding algorithm (FISTA), we introduce a learned momentum scheme to achieve "optimal" gain. Extensive encoding and decoding results confirm our hypothesis and prove the effectiveness of our method.

1. Introduction

Consider the constrained minimization problem for a smooth function $f : \mathbb{R}^m \to \mathbb{R}$ and a convex function $g : \mathbb{R}^n \to \mathbb{R}$ which may be non-differentiable:

$$\arg\min_x f(x) + g(x). \tag{1}$$

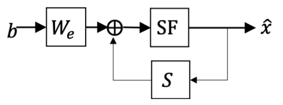
If $f(x) = \frac{1}{2}\|b - Ax\|_2^2$ and $g(x) = \lambda\|x\|_1$, where $b \in \mathbb{R}^m, A \in \mathbb{R}^{m\times n}$ with $m \ll n$, $x \in \mathbb{R}^n$, λ is a positive scalar that balances the strength between regularized least-squares and the regularization. This minimization problem is called the LASSO regression and is the central issue in this paper.

In recent years, variants of this problem find practical relevance in numerous disparate fields, such as computer vision [8], image compressive sensing [29], and sparse subspace clustering [11], all of them relying heavily on the resolution of ℓ_1-regularized least squares. However, the processing is usually computationally expensive, thus rendering most existing ℓ_1 encoders less effective in practice. For instance, when it comes to representation-based face recognition [26], it always costs few tens of iterations using proximal gradient methods to calculate the sparse representation. Therefore, the acceleration for ℓ_1-solvers is a critical factor in real-time applications.

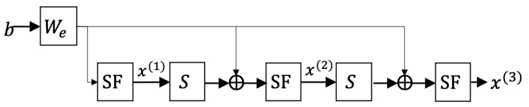
Iterative shrinkage thresholding algorithm(ISTA, [9]), as an extension of the proximal gradient method, is one of the most well-known schemes to approach this issue. Later, by binding to it "Nesterov acceleration", a novel fast iterative shrinkage thresholding algorithm (FISTA, [4]) was proposed, and achieved improved complexity results of $O(\frac{1}{k^2})$ compared to $O(\frac{1}{k})$ for ISTA, where k is the number of iterations. The approximate message passing (AMP, [10]) algorithm was applied to solve the problem. Other schemes [3, 17] also showed their effectiveness.

However, the past decade has seen a raising general interest in implementing these iterative algorithms as DNNs, i.e., taking traditional iterative algorithms into data-driven learning mechanisms to obtain accelerated sparse solvers, which we focus on in this paper. LISTA [14] shows a representational way of unfolded iterative algorithms as neural networks. It only needs few iterations to achieve comparative results with the baseline algorithm ISTA. Following LISTA and its usefulness and versatility, a variety of other algorithms, such as LISTA-based or its variants [21, 15, 20, 6, 7, 2, 16, 1], have been developed. At the same time, many other extensions have been put forward to solve other minimization problems. For example, [24, 28] researched the ℓ_0-based sparse approximation and [25] investigated the ℓ_∞-constrained representation by unrolling the alternating direction method of multipliers (ADMM) algorithm.

In this paper, we pursue a lightweight and cost-effective learning-based ISTA to produce as low as possible error for sparse solutions in a given number of iterations. First of all, we propose training ISTA and tuning its step size. Our strategy makes ISTA trainable by using back-propagation so that it not only fits the sparse dictionary on a training



(a) RNN structure of ISTA.



(b) Diagram of LISTA.

Figure 1: Block diagrams of ISTA and LISTA. In the block diagram of LISTA, we only show three iterations here, and $x^{(0)}$ is a zero vector.

dataset, but also improves the speed of convergence. Our novelty resides in the use of back-propagable eigen decomposition in unfolded ISTA to enable the tuning of step sizes. Then, in order to improve its effectiveness, we replace step sizes tuning by a learned approach and, to reduce the number of trainable parameters in this scheme, we use an existing ”tied” weights design from [5] to rebuild our algorithms. Finally, we re-investigate the momentum term introduced in FISTA, which is computed by taking the special scalar times the difference of the last two outputs, with the hypothesis of the existence of a data-dependent optimum for the special scalar. We suggest learning the scalar independently of the layer at index k. The basic idea of adding the ”momentum” term into learning-based algorithms has been proposed by other researchers ([19, 30, 27]). However, unlike in these researches, where adding momentum items usually cause the number of trainable parameters to increase a lot in some cases, our contribution is to study an ”optimal” scalar for the momentum which adds an additional parameter but helps to achieve both ”good” performances and ”low” consumption of computational resources.

Outline This paper begins with a survey on the unfolding of iterative algorithms. The specific algorithms descriptions are made in section 3. Our numerical experiences are shown in section 4 and the conclusion is found in section 5.

2. Related works

ISTA[9], the baseline algorithm is written as:

$$x^{(k+1)} = SF\left(x^{(k)} + \alpha A^T\left(b - Ax^{(k)}\right), \alpha\lambda\right). \quad (2)$$

Here, $SF : \mathbb{R}^n \to \mathbb{R}^n$ is a shrinkage operator defined as $SF(x,\theta) = sign(x)max(|x| - \theta, 0)$. The positive constant α is the step size, of which the optimal value is analytically known to be $\frac{1}{L}$, i.e., the reciprocal of the smallest Lipschitz constant L of $\bigtriangledown f$. L is usually taken as the largest eigenvalue of A^TA. LISTA [14] remodeled ISTA as a recursive structure presented in Figure 1a, and further implemented it into a truncated form that is illustrated in Figure 1b, thus they rewrote Equation 2 as:

$$x^{(k+1)} = SF\left(W_e b + Sx^{(k)}, \theta\right), \quad (3)$$

where $W_e = \frac{A^T}{L}, S = I - \frac{A^TA}{L}, \theta = \frac{\lambda}{L}$. The trainable parameters of LISTA are $\{(W_e, S, \theta)\}$.

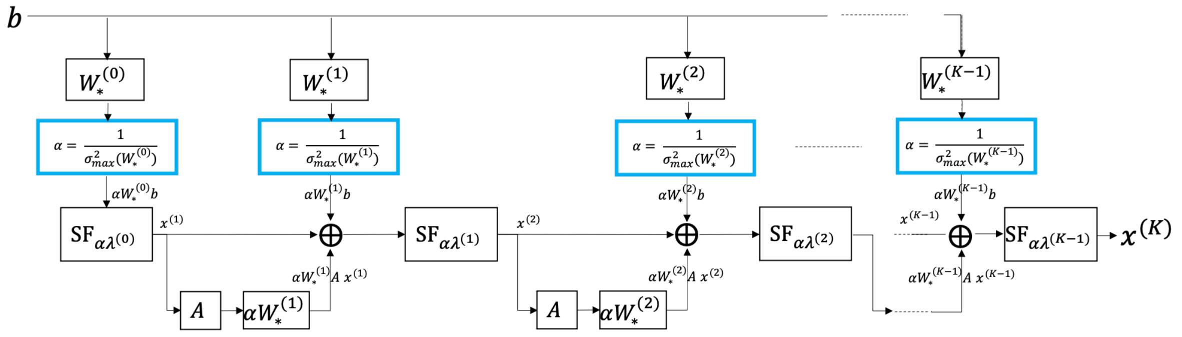
Discussing the weights structure, an unshared and partial weight coupling algorithm LISTA-CP [7] was proposed. In LISTA-CP, they pointed out that $S^{(k)} - (I - {W_e}^{(k)}A) \to 0$ and $\theta^{(k)} \to 0$ as k increases under certain conditions. [18] decomposes ${W_e}^{(k)}$ as the product of a scalar $\eta^{(k)}$ and a matrix W_e independent of layer index k, where W_e has small coherence with A. Furthermore, they get rid of the heavy matrix learning by using a pre-computed ”good” weight $\tilde{W}$, which helps to reduce the numbers of trainable parameters effectively. Using the LISTA framework, [2] proposed a learning step sizes scheme, [5] extended it to AMP [10], [19] modified FISTA and proposed a learned FISTA (LFISTA) but with more trainable parameters. Combining ISTA with the long short term memory(LSTM), [30] proposed a novel RNN based solver which helps to update the parameters and encapsulates the historical information. More recently, the ”gate” mechanism based algorithm GLISTA ([27]) helped compensating the insufficient gain of code components in sparse solutions.

Extending from theoretical innovations to application expansions, in more recent years, trainable sparse coding algorithms have received widespread attention in the fields of image and signal processing. Various approaches [19, 13, 7] have been put forward to unveil the theory behind LISTA. In practice, [29] developed a novel ISTA-Net, which is an extension of ISTA, used to solve the CS reconstruction problem. A growing body of algorithms for extending approximate sparse coding models to CSC (convolutional sparse coding) models has been proposed [23, 22, 18]. By replacing the regular dictionary with a convolutional one, [22] rewrites ISTA as a solution to the CSC problem, and both the sparse coding and dictionary are learned in this framework. Their innovative works showed that learned CSC is efficient for processing the whole image and showed impressive performances compared to other sparse coding based techniques on images inpainting and denoising.

3. Method

3.1. Step sizes schemes for unfolded ISTA

Tuned step sizes LISTA-CP [7] shows that using a partial weight coupling structure can help to reduce the number of trainable parameters in unfolded ISTA. This structure is the one we consider next. In this partial weight coupling structure, the step size is part of the initialized value for the matrix ${W_e}^{(k)} = \alpha A^T$ and the threshold $\theta^{(k)} = \alpha\lambda$. However, the step size in the original ISTA was a critical point


(a) The step size tuning method. The blue block is the scheme for the automatic computation of the step size α using SVD. $\sigma^2_{max}\left(W_*\right)$ is the largest singular square value of W_*. The computed step size is also used for the soft shrinkage operator.

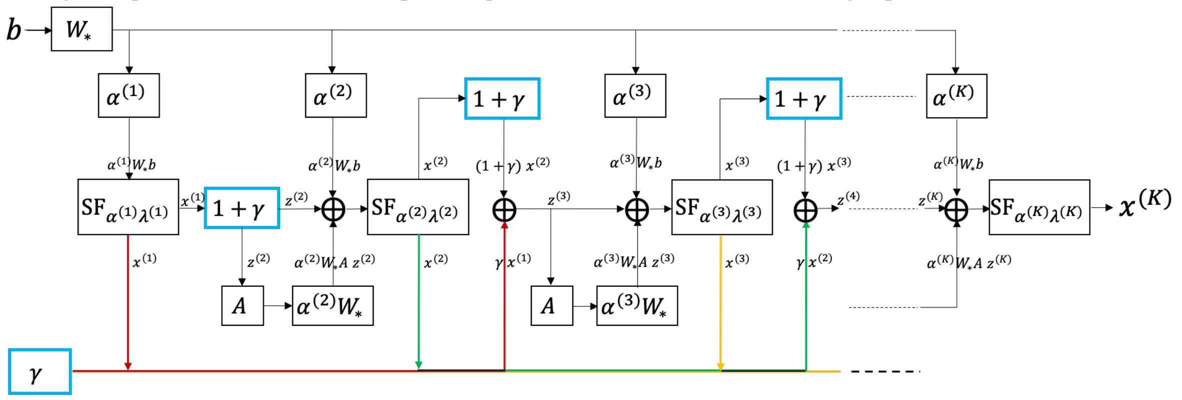

(b) The diagram of the step size learning method combined with the learned momentum scheme (i.e., LM-FISTA algorithm). The blue block denotes the usages of learned scalar γ. $z^{(1)} = x^{(0)} = 0$. The red line indicates that the previous solution $x^{(1)}$ times γ as part of the inputs to the addition operation, so do the other colored lines, we use different colored lines to indicate these operations are processed in sequence.

Figure 2: Block diagram for our implemented K-layer models.

in the approximation of ground truth sparse solutions. If the step size was too small, the number of iterations, or the number of layers after unfolding ISTA, would be very large. Conversely, if the step size was too large, or the Lipschitz constant L was too small, ISTA might be unstable and diverge. It turns out that a suitable and minimal Lipschitz constant L would be incredibly important in speeding-up estimations of sparse solutions. Normally, the "optimal" step size is used as part of the initial value for the trainable weights in unfolded iterative algorithms. In order to keep the "best" step size to achieve better performances, we propose to keep the step size α to be the reciprocal of the analytically optimal largest singular square value $\sigma^2_{max}(A^T)$, with the dictionary A^T being trainable. We fine-tune step sizes using SVD after dictionary learning and leverage the sparsity by making the scalar λ learned. In our scheme, the matrix A^T manifests as W_*:

$$\begin{aligned} v^{(k)} &= b - Ax^{(k)}, \\ x^{(k+1)} &= SF\left(x^{(k)} + \alpha {W_*}^{(k)} v^{(k)}, \lambda^{(k)}\alpha\right). \end{aligned} \tag{4}$$

The trainable parameters are $\Theta = \{({W_*}^{(k)}, \lambda^{(k)})\}_{k=0}^{K-1}$. Step size α is automatically calculated during training. We name this scheme tuned steps LISTA. The block diagram of the method is represented in Figure 2a.

Learned step sizes [2] proposed a strategy to learn the step size for unfolded sparse coding, they use step size to be the only trainable parameter in learned ISTA and train the network with an unsupervised objective:

$$\begin{aligned} v^{(k)} &= b - Ax^{(k)}, \\ x^{(k+1)} &= SF\left(x^{(k)} + \alpha^{(k)} A^T v^{(k)}), \alpha^{(k)}\lambda\right). \end{aligned} \tag{5}$$

The initial value for $\alpha^{(k)}$ is $\frac{1}{L}$. Their results showed that the learned step size scheme is better than other works when solutions are sparse enough in an unsupervised context. This means it is effective to train the step size independently if we fine-leverage the scalar λ to be sparse enough.

Our tuned step sizes scheme preserves the "best" step size can accelerate the speed of approaching ground truth, however, the results of the current study are limited by the time-consuming SVD computations. To reduce such computations while training models, we propose to learn the step size instead of automatically computing it with the SVD. Thus, the learned step size version is returned as:

$$\begin{aligned} v^{(k)} &= b - Ax^{(k)}, \\ x^{(k+1)} &= SF\left(x^{(k)} + \alpha^{(k)} {W_*}^{(k)} v^{(k)}, \lambda^{(k)}\alpha^{(k)}\right). \end{aligned} \tag{6}$$

The trainable parameters are $\Theta = \{({W_*}^{(k)}, \alpha^{(k)}, \lambda^{(k)})\}_{k=0}^{K-1}$. We name this scheme as learned steps LISTA. As described in [2], they consider a LISTA layer as a function $\varphi_\theta : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ parameterized by $\Theta_k = \{W, \alpha, \beta\} \in \mathbb{R}^{m\times n} \times \mathbb{R}_*^+ \times \mathbb{R}_*^+$:

$$\varphi_\theta(x, b) = SF\left(x + \alpha W^T \left(b - Ax^{(k)}\right), \beta\lambda\right). \tag{7}$$

Taking $W = A, \alpha = \beta = \frac{1}{L}$ outputs the same results as one iteration of ISTA. In this context, we consider a LISTA layer as a function φ_θ parameterized by $\Theta_k = \{W, \alpha, \lambda\}$.

Tied approach for learned steps LISTA The weights design of learned iterative algorithms is directly related to the number of trainable parameters. Two types of weights structures are normally discussed in the field, i.e., shared and unshared weights designs. For a K-layer model, shared weights indicate that there is only one trainable parameter for every single weight, that is, weights independent of layer index k, and this parameter is trained through every layer. If there are K trainable parameters for every single weight, and these parameters are trained independently in the corresponding layer, then it is unshared weights. The weights of LISTA were originally proposed as a shared structure in [14]. [7] introduced a unshared weights diagram of LISTA. [19] showed that learning shared layer weights is less effective and thus also used unshared weights in their own algorithm LFISTA. Learning unshared layer weights has shown better performances than shared ones in approximating sparse solutions. However, it results in too many trainable parameters as the number of iteration (or layer) increases. The tied structure which changed the trainable parameters for LISTA as $\Theta = \{W_e, S, \{\theta^{(k)}\}_k^{K-1}\}$ was introduced in [5]. It reduces their numbers and achieves better performances compared to unshared LISTA and shared LISTA as showed in our experiments (Figure 5). [18] also used such idea. Since our experiments also show that tied weights structure is effective, hereafter we also adopt this tied structure in our scheme:

$$\begin{aligned} v^{(k)} &= b - Ax^{(k)}, \\ x^{(k+1)} &= SF\left(x^{(k)} + \alpha^{(k)} W_* v^{(k)}, \lambda^{(k)}\alpha^{(k)}\right). \end{aligned} \tag{8}$$

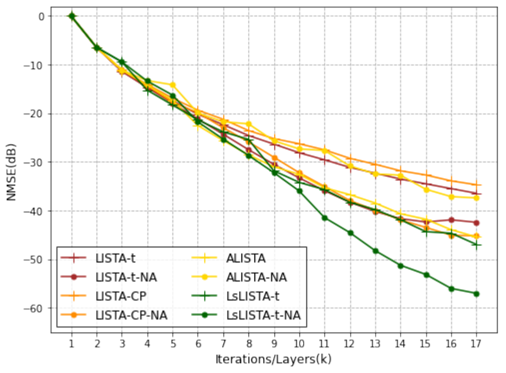


Figure 3: Validation of learned ISTA with "Nesterov acceleration", $\lambda = 0.4$,SNR = ∞. Here "-NA" means the original algorithm with "Nesterov acceleration", "-t" means the original algorithm with tied weights structure.

The trainable parameters become $\Theta = \{W_*, \{(\alpha^{(k)}, \lambda^{(k)})\}_{k=0}^{K-1}\}$. We name this scheme as learned steps LISTA with tied weights. We notice that our scheme has a similar trainable parameters setting to TiLISTA([18]), but in TiLISTA they set ${W_e}^{(k)} = \eta^{(k)} W_e$ and initialized W_e as $\frac{A^T}{L}$ and $\eta^{(k)}$ to 1. Ours is from the objective of learning step sizes and W_* is initialized as A^T and $\alpha^{(k)}$ is initialized as $\frac{1}{\sigma_{max}^2(A^T)}$.

3.2. Learning of momentum for unfolded FISTA

[27] speculated that the element-wise update of (L)ISTA "lags behind" the optimal solution so it is suggested that overshoots are needed to reach the optimum. They acted on the sparse solutions by using a "gate" scheme: extra functions are added to enlarge the code components of sparse estimations of the learned ISTA. They introduced the "gain gate" and the "overshoot gate" in their algorithms to achieve better performances. To get $x^{(k+1)}$, the "gain gate" is acting on $x^{(k)}$ before the soft shrinkage thresholding, and the "overshoot gate" acts on the solution after the operator and the last solution $x^{(k)}$. In their "overshoot gate" strategy, they made the "gate" to be a time and input-varying function. We argue that adding extra functions can be both time and computationally consuming. Moreover, it makes the number of trainable parameters to grow too much under some of their settings. At the same time, their way to combine two functions by using them in different iterations or layers also makes the algorithm complicated to implement in a handcrafted way. To enlarge the components of the sparse estimations, we prefer to directly extend FISTA [4].

Here, we take a view to make FISTA into a learned algorithm following the idea of LISTA. [19] introduced a learned FISTA algorithm, however, it turns out that it has too many trainable parameters compared to other learned it-

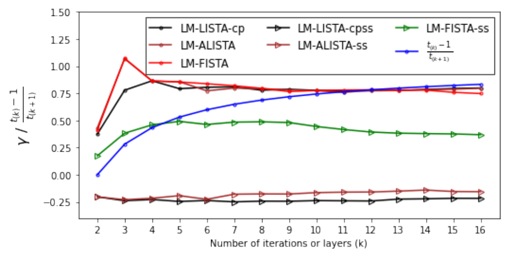


Figure 4: Validations of γ, the blue line labels the special scalar $\frac{t_{(k)}-1}{t_{(k+1)}}$ in FISTA. Here, ”LM-” means learned momentum scheme is added, ”ss” denotes the ”support selection”([7]) scheme is used in the algorithm.

erative algorithms such as LISTA [14] or LISTA-CP [7] and so on. And for each layer, because the designed weights for LFISTA include a special linear combination that is calculated in a handcrafted way, it is hard for them to simplify their trainable parameters to pursue a lightweight structure like LISTA with tied weights. Still, LFISTA has shown the effectiveness of unrolling FISTA into a learned algorithm. Combining the FISTA algorithm with the learned strategies, we decompose learned FISTA into two parts: the learned algorithms and the ”Nesterov acceleration” described in FISTA. We take the tied weights of LISTA as an example to illustrate it with ”Nesterov acceleration”:

$$z^{(1)} = x^{(0)}, t_{(1)} = 1, \tag{9}$$

$$x^{(k)} = SF\left(W_e b + S z^{(k)}, \theta^{(k)}\right), \tag{10}$$

$$t_{(k+1)} = \frac{1}{2}\left(1 + \sqrt{1 + 4t_{(k)}^2}\right), \tag{11}$$

$$z^{(k+1)} = x^{(k)} + \frac{t_{(k)} - 1}{t_{(k+1)}}\left(x^{(k)} - x^{(k-1)}\right). \tag{12}$$

Meanwhile, we can also take the same steps, add ”Nesterov acceleration” into LISTA-CP [7], ALISTA [18] and LsLISTA-t. From Figure 3, comparing the ”-NA” schemes with the original ones, LISTA-t-NA, LISTA-CP-NA and LsLISTA-t-NA show better performances than their original counterparts, and LsLISTA-t-NA performs the best as the number of layers or iterations increase. However, ALISTA-NA fails to perform better than ALISTA or even becomes worse. We speculate that adding layer-fixed momentum to the pre-calculated ”good” weight scheme is not that adaptive and sufficient, and thus it causes ALISTA to need more iterations or layers to achieve better performance. Under this conjecture, a more powerful momentum is needed for ALISTA to accelerate the sparse solver.

According to the discussion above, we can then argue whether the added ”Nesterov acceleration” for the other algorithms discussed above is enough or not, and if there exists a data-dependent ”optimal” momentum after training. In order to get fast convergence of the original FISTA, it is well known that one needs to reset $t_{(k)}$ to 0 after a few tens iterations k. In this handcrafted way, the special scalar $\frac{t_{(k)}-1}{t_{(k+1)}}$ is following a monotonically increasing sequence between 0 and 1 as shown in Figure 4. To get rid of this handmade operation and gain enough momentum for ISTA, we introduce a strategy for learning the ”momentum” by taking the special layer-fixed scalar as a learned one and making it to be trainable and adaptive. We combine our learned steps LISTA with tied weights scheme and the learned scalar to rewrite learned FISTA as (block diagram presented in Figure 2b):

$$z^{(1)} = x^{(0)}, v^{(k)} = b - Az^{(k)}, \tag{13}$$

$$x^{(k)} = SF\left(z^{(k)} + \alpha^{(k)} W_* v^{(k)}, \lambda^{(k)}\alpha^{(k)}\right), \tag{14}$$

$$z^{(k+1)} = x^{(k)} + \gamma\left(x^{(k)} - x^{(k-1)}\right). \tag{15}$$

The trainable parameters are $\Theta = \{W_*, \gamma, \{\alpha^{(k)}, \lambda^{(k)}\}_{k=0}^{K-1}\}$. The initial value of γ is 0, γ is a shared scalar and is learned as long as $k > 1$. The shared scalar γ helps us to observe the tendency of ”optimal” acceleration from the 1-layer model to the K-layer model. Under our hypothesis, γ is learned and tends to reach a stable ”optimum” point between 0 and 1 as the number of layers or iterations increase.

Validation of γ Explanation for abbreviations mentioned below are put in section 4. We show the values of γ and the special scalar $\frac{t_{(k)}-1}{t_{(k+1)}}$ after training of the 1-layer model to K-layer model in Figure 4, K=16 in our setting. $\frac{t_{(k)}-1}{t_{(k+1)}}$ keeps increasing as iterations or layers increase, but the speed to acquire the ”appropriate” value is still slow . ”LM” tagged and without ”ss” labeled algorithms show that γ tends to be stable and below 1 as k increases. This means there exists a data-dependent optimal γ after training for ”LM” tagged algorithms. However, when it comes to the algorithms that both ”LM” and ”ss” tagged, only our algorithm LM-FISTA-ss is able to learn a ”good” γ. We notice that the ”support selection” scheme and our ”learned momentum” both aim to help getting better sparse solutions selections, the ”support selection” takes effect before the shrinkage operator and ours after it. ”LM-LISTA-cpss” and ”LM-ALISTA-ss” have negative γ, we think this is because in these two algorithms, after the ”support selection” design, sparse solutions already got their best ”selected” components and the learned ”momentum” seems to have a negative effect on them resulting in an over selection of components. Moreover, though the empirical scheme ”support selection” has shown its effectiveness, it still has two hyperparameters that need to be manually tuned. Our learned momentum is more adapted to this situation and can achieve better performance as it keeps learning a stable and ”good” value for the acceleration speed of approximated solutions.

LISTA-t [5]	LISTA-CPss [7]	ALISTA-ss [18]	**TsLISTA**	**LsLISTA-t**	**LM-FISTA**
$O(mn+n^2+K)$	$O(Kmn+K)$	$O(K)$	$O(Kmn+K)$	$O(mn+K)$	$O(mn+K)$

Table 1: Comparison of the number of trainable parameters for learned iterative algorithms. To clarify, algorithm LFISTA([19]) is $O(Kn^2+Kn^2+Kmn+K)$, and GLISTA([27]) depend on their added "gate" functions and LISTA or LISTA-CP algorithms. LM-ALISTA is $O(K)$, and "LM-" tagged algorithms only need to add one parameter.

As mentioned before, we pursue a lightweight learn-based algorithm, the numbers of trainable parameters are essential to evaluate the simplicity of the DNNs models. Here, we describe the different variants of LISTA and the number of trainable parameters (presented in Table 1).

4. Numerical results

We present all the simulations and practical experiments in this section.

Notation For simplicity, we use TsLISTA to denote the tuned step sizes LISTA, LsLISTA stands for learned step sizes LISTA, to specify, LM-FISTA refers to LsLISTA-t combined with the "learned momentum" scheme. LISTA always refers to the baseline LISTA [14], LISTA-us means the unshared weights structure is used for LISTA. Moreover, "-t" is the abbreviation for "tied" and it means tied structure is used in algorithm, "LM-" is the abbreviation for "learned momentum" scheme and it represents the scheme is added. The abbreviation "ss" is used for the added "support selection" ([7]) scheme, for example, ALISTA-ss means the "support selection" is added to ALISTA [18]. As well, "-NA" is the abbreviation for "Nesterov acceleration".

4.1. Simulation experiments

Experiments setting The shape of A is m=250, n=500. A is sampled from a standard Gaussian distribution, $A_{ij} \sim N(0, \frac{1}{m})$. For the ground truth sparse vectors x^*, non-zero entries are 10% of n, each non-zero entry is sampled from a standard Gaussian distribution and the sample assumes a uniform distribution over all entries, $\lambda = 0.4$. Training batch size is 64, validation data is 1000, test data 1000, optimizer is Adam. For every different noise setting, we have different test data, but for every algorithm, they are tested with the same test data. The data is generated in an ideal setting, meaning we use synthesized data that abide by the same distribution and the number of training samples grows as the training proceeds. It is used for training with different learning rates settings, following the previous works([7, 18]). We use TensorFlow-GPU 2.2.0 in our experiments. We evaluate the algorithms with NMSE([7]):

$$NMSE\left(\hat{x}, x^*\right) = 10log_{10}\left(\frac{\mathbb{E}\left\|\hat{x}-x^*\right\|_2^2}{\mathbb{E}\left\|x^*\right\|_2^2}\right). \quad (16)$$

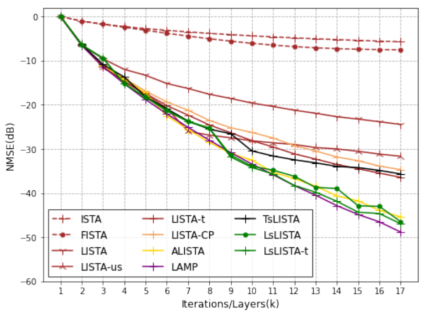
Two groups of simulations were created to evaluate our algorithms. The first group is simulated in SNR(Signal-to-Noise Ratio)=∞ and 20. In the second group, four validation results are presented, the first validation is simulated in SNR=20, and the rest of them are simulated in SNR = ∞, 20, and 10 respectively.

Verification group 1 In the first group, we verify the tuning and learning of step sizes for unfolded ISTA (see Figure 5). For tuning of step sizes, TsLISTA has the same number of trainable parameters as LISTA-CP [7] but shows better performances, and it also shows comparable ones to LISTA-t, the tuned step sizes scheme is acceptable here. As for the learned step sizes schemes LsLISTA and LsLISTA-t, they show close results and performs better compared to ALISTA [18] as layer or iteration increases, thus validating our approach. Comparing three weights designs for LISTA, that is, LISTA [14], LISTA-us [7] and LISTA-t [5], the tied one proved to be the best choice. Still LsLISTA-t has fewer trainable parameters than LsLISTA, making it a better option.

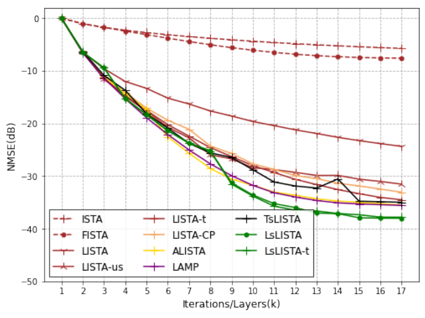
Verification group 2 Four validation results are included in group 2, we divide them into two parts. In the first part, we verify that our learned scalar γ is more effective than the handcrafted "Nesterov acceleration" in Figure 6a, all "LM-" labeled algorithms are better than their baseline algorithm counterpart. Especially when it comes to ALISTA [18], proving our hypothesis in section 3.2 that a learned "good" scalar γ helps to gain powerful momentum to speed-up the approximation of sparse solutions. Comparison results of our algorithms with state-of-the-art ones are shown in Figure 6b,6c and 6d. Our scheme LM-FISTA-ss performances are good, as expected, whether it is under noiseless or noisy cases. ALISTA-ss and LsLISTA-t-ss perform comparably when SNR=∞, but LsLISTA-t-ss shows better as SNR becomes lower. Although "LM" tagged algorithms are worse than "ss" tagged ones, learned momentum still has its utility as "support selection" can suffer from its manual weak point that we mentioned before.

4.2. Classification experiments

Experiments setting The solutions of sparse representation-based face recognition([26])are naturally sparse, making this problem suited to our case so we choose it to demonstrate a use-case for our algorithms. We use the Extended Yale B database [12] to realize this

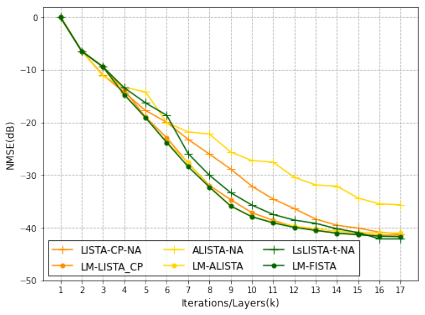


(a) SNR = ∞.

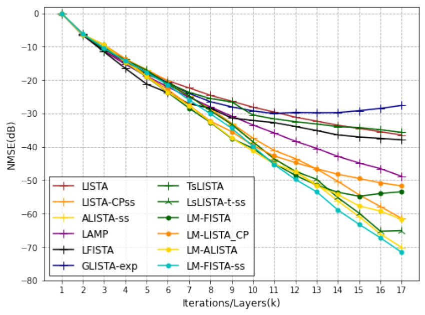


(b) SNR = 20.

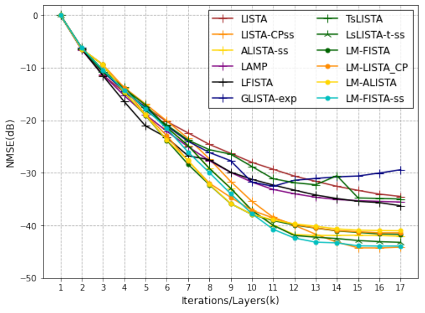
Figure 5: Validation group 1, λ = 0.4.



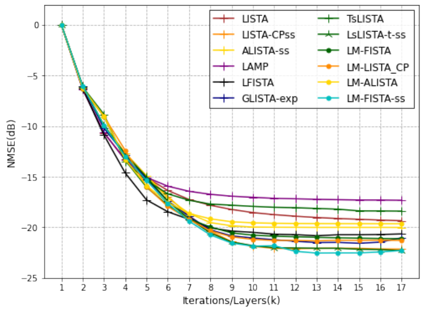
(a) SNR = 20



(b) SNR = ∞



(c) SNR = 20



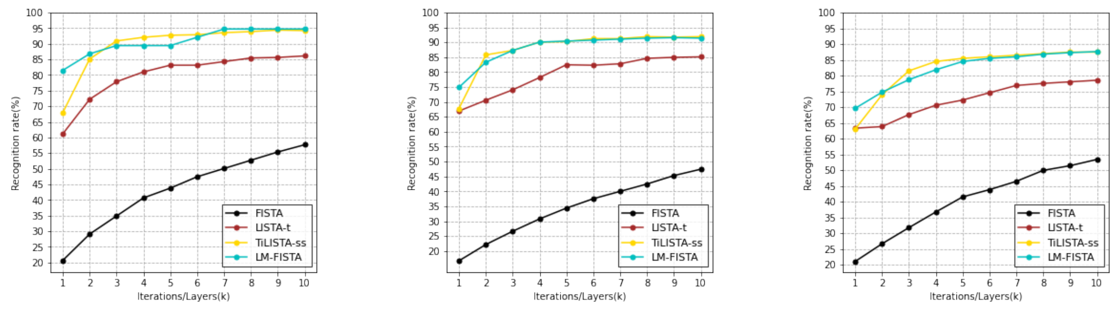
(d) SNR = 10

Figure 6: Validation group 2, $\lambda = 0.4$. The algorithm GLISTA [27] we use the "-exp" tag means that we use the "exponential function" scheme ([27]) here as an example for GLISTA. Because "-exp" shows good performances in their work, to simplify the discussion, we choose this one as its representation.

classification experiment. The database includes 2414 frontal-face images of 38 individuals. All pictures are captured under different lighting setting for each individuals. The shape of each cropped picture is 192×168. Following the setting in [26], we use a random matrix R, sampled from a standard Gaussian distribution to generate the feature space, setting them to 56, 120 and 504. The input $\tilde{b}$ is manifested as $\tilde{b} = Rb$, the dictionary $\tilde{A}$ is $\tilde{A} = RA$, m=56, 120 and 504 respectively, $n = 1205$. Then the ℓ_1 minimization problem can be written as:

$$\tilde{x}_1 = \arg\min_{x} \|x\|_1 \quad \text{s.t.} \quad RAx = \tilde{b}. \tag{17}$$

We randomly select half of the images from each subjects to build our dictionary A, each column of $\tilde{A}$ is normalized. The training dataset size is 1205, validation dataset size is 604 and test dataset size is 605, $\lambda = 0.4$. We use the same training strategy as in the simulation experiment mentioned previously. We use the SRC scheme [26] for the classification part but solve the ℓ_1 minimization problem by using different learned iterative algorithms. The loss function is



(a) Features dimensions:504 (b) Features dimensions:120 (c) Features dimensions:56

Figure 7: Validation of classification experiments for three feature spaces: 504, 120 and 56.

Features	Recognition Rate(%)			
	FISTA [4]	LISTA-t [5]	TiLISTA-ss [18]	**LM-FISTA**
504	91.07	86.12	94.38	**94.71**
120	89.75	85.12	**91.90**	91.57
56	85.78	78.51	87.43	**87.60**

Table 2: Validation of classification experiments, "Features" stands for features dimensions. For FISTA, 100 iterations are needed for these results, for the other algorithms, we got them after 10 iterations.

defined as:

$$L(\Theta) = \frac{1}{N}(L_1(\Theta) + L_2(\Theta)) \tag{18}$$

$$L_1(\Theta) = -\sum_{i=0}^{N-1} \tilde{y}_i \log\left(b{*}_i^k(\Theta)\right) \tag{19}$$

$$L_2(\Theta) = \sum_{i=0}^{N-1} \left\|\tilde{b} - \tilde{A}x_i^K(\Theta)\right\|_2^2 + \lambda \left\|x_i^K(\Theta)\right\|_1 \tag{20}$$

Where $\tilde{y}_i$ indicates the class label, and $b{*}_i^k(\cdot)$ is the predicted vector consisting of the possibilities for each class.

Validation of classification ALISTA-ss [18] seems to fail in dealing with this problem, so we use another implemented scheme named TiLISTA-ss instead. FISTA and LISTA are also included for comparison. Figure 7 shows the specific recognition rate as the number of iteration or layer increase. After just one iteration, our algorithm LM-FISTA achieved 81.49%, 74.88%, and 69.59% accuracy when the features dimensions are 504, 120 and 56 respectively. However, LISTA-t needs 5, 3 and 4 iterations to show the same performance. FISTA needs around 30 for these three cases. In Table 2, we can see that FISTA needs 100 iterations to get acceptable results but still not as good as ours, indeed, for the first 10 iterations, LM-FISTA achieved 94.71% accuracy when the input feature size is 504 and 87.60% when the size is 56. This classification application shows the effectiveness of our algorithm, showing better performances than others presented here.

5. Conclusion

The purpose of this paper was to find a lightweight way for trainable sparse coding to speed-up the approximation of "ground truth" sparse solutions. We reaffirmed the importance of step size in learned ISTA with the ideas of step size tuning and step size learning, then experimentally demonstrated the hypothesis of finding a data-dependent "optimal" scalar for the "momentum" of learned ISTA, achieving great performances in our simulation experiments as well as reducing consumption of computational resources. We showed that our schemes are also suited for classification applications such as representation-based face recognition. Future works can include explaining our assumptions in a theoretical way and putting more insights on solving the sparse representation-based face recognition by using unfolded iterative algorithms.

References

[1] Aviad Aberdam, Alona Golts, and Michael Elad. Ada-lista: Learned solvers adaptive to varying models. *arXiv preprint arXiv:2001.08456*, 2020.

[2] Pierre Ablin, Thomas Moreau, Mathurin Massias, and Alexandre Gramfort. Learning step sizes for unfolded sparse coding. *arXiv preprint arXiv:1905.11071*, 2019.

[3] Manya V Afonso, José M Bioucas-Dias, and Mário AT Figueiredo. Fast image recovery using variable splitting and constrained optimization. *IEEE transactions on image processing*, 19(9):2345–2356, 2010.

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

[5] Mark Borgerding and Philip Schniter. Onsager-corrected deep learning for sparse linear inverse problems, 2016.

[6] Mark Borgerding, Philip Schniter, and Sundeep Rangan. Amp-inspired deep networks for sparse linear inverse problems. *IEEE Transactions on Signal Processing*, 65(16):4293–4308, 2017.

[7] Xiaohan Chen, Jialin Liu, Zhangyang Wang, and Wotao Yin. Theoretical linear convergence of unfolded ista and its practical weights and thresholds. *arXiv preprint arXiv:1808.10038*, 2018.

[8] Adam Coates and Andrew Y Ng. The importance of encoding versus training with sparse coding and vector quantization, 2011.

[9] Ingrid Daubechies, Michel Defrise, and Christine De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.

[10] David L Donoho, Arian Maleki, and Andrea Montanari. Message-passing algorithms for compressed sensing. *Proceedings of the National Academy of Sciences*, 106(45):18914–18919, 2009.

[11] Ehsan Elhamifar and René Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2765–2781, 2013.

[12] Athinodoros S. Georghiades, Peter N. Belhumeur, and David J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE transactions on pattern analysis and machine intelligence*, 23(6):643–660, 2001.

[13] Raja Giryes, Yonina C Eldar, Alex M Bronstein, and Guillermo Sapiro. Tradeoffs between convergence speed and reconstruction accuracy in inverse problems. *IEEE Transactions on Signal Processing*, 66(7):1676–1690, 2018.

[14] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding, 2010.

[15] John R Hershey, Jonathan Le Roux, and Felix Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*, 2014.

[16] Daisuke Ito, Satoshi Takabe, and Tadashi Wadayama. Trainable ista for sparse signal recovery. *IEEE Transactions on Signal Processing*, 67(12):3113–3125, 2019.

[17] Yingying Li and Stanley Osher. Coordinate descent optimization for l 1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems & Imaging*, 3(3):487, 2009.

[18] Jialin Liu, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Alista: Analytic weights are as good as learned weights in lista, 2019.

[19] Thomas Moreau and Joan Bruna. Understanding trainable sparse coding via matrix factorization. *arXiv preprint arXiv:1609.00285*, 2016.

[20] Pablo Sprechmann, Alexander M Bronstein, and Guillermo Sapiro. Learning efficient sparse and low rank models. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1821–1833, 2015.

[21] Pablo Sprechmann, Roee Litman, Tal Ben Yakar, Alex Bronstein, and Guillermo Sapiro. Efficient supervised sparse analysis and synthesis operators. *Advances in Neural Information Processing Systems (NIPS)*, pages 908–916, 2013.

[22] Hillel Sreter and Raja Giryes. Learned convolutional sparse coding, 2018.

[23] Bahareh Tolooshams, Sourav Dey, and Demba Ba. Scalable convolutional dictionary learning with constrained recurrent sparse auto-encoders, 2018.

[24] Zhangyang Wang, Qing Ling, and Thomas Huang. Learning deepℓ_0 encoders, 2016.

[25] Zhangyang Wang, Yingzhen Yang, Shiyu Chang, Qing Ling, and Thomas S Huang. Learning a deep ℓ_∞ encoder for hashing. *arXiv preprint arXiv:1604.01475*, 2016.

[26] John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, 31(2):210–227, 2008.

[27] Kailun Wu, Yiwen Guo, Ziang Li, and Changshui Zhang. Sparse coding with gated learned ista, 2019.

[28] Bo Xin, Yizhou Wang, Wen Gao, and David Wipf. Maximal sparsity with deep networks? *arXiv preprint arXiv:1605.01636*, 2016.

[29] Jian Zhang and Bernard Ghanem. Ista-net: Interpretable optimization-inspired deep network for image compressive sensing, 2018.

[30] Joey Tianyi Zhou, Kai Di, Jiawei Du, Xi Peng, Hao Yang, Sinno Jialin Pan, Ivor Tsang, Yong Liu, Zheng Qin, and Rick Siow Mong Goh. Sc2net: Sparse lstms for sparse coding, 2018.