L_p-Norm Constrained Coding With Frank-Wolfe Network

Ke Sun, Zhangyang Wang, Member, IEEE, Dong Liu, Member, IEEE, Runsheng Liu, and Zheng-Jun Zha

Abstract—We investigate the problem of L_p-norm constrained coding, i.e., converting signal into code that lies inside an L_p-ball and most faithfully reconstructs the signal. While previous works known as sparse coding have addressed the cases of L_0 and L_1 norms, more general cases with other p values, especially with unknown p, remain a difficulty. We propose the Frank-Wolfe Network (F-W Net), whose architecture is inspired by unrolling and truncating the Frank-Wolfe algorithm for solving an L_p-norm constrained problem. We show that the Frank-Wolfe solver for the L_p-norm constraint leads to a novel closed-form nonlinear unit, which is parameterized by p and termed pool_p. The pool_p unit links the conventional pooling, activation, and normalization operations, making F-W Net distinct from existing deep networks either heuristically designed or converted from projected gradient descent algorithms. We further show that the hyper-parameter p can be made learnable instead of pre-chosen in F-W Net, which gracefully solves the L_p-norm constrained coding problem with unknown p. We evaluate the performance of F-W Net on an extensive range of simulations as well as the task of handwritten digit recognition, where F-W Net exhibits strong learning capability. We then propose a convolutional version of F-W Net, and apply the convolutional F-W Net into image denoising and super-resolution tasks, where F-W Net all demonstrates impressive effectiveness, flexibility, and robustness.

Index Terms—Convolutional network, Frank-Wolfe algorithm, Frank-Wolfe network, L_p-norm constrained coding.

I. INTRODUCTION

A. L_p-Norm Constrained Coding

Assuming that there are a set of vectors \{x_i \in \mathbb{R}^n | i = 1, 2, \ldots, N\}, we want to encode each vector x_i into code z_i \in \mathbb{R}^m, such that the code reconstructs the vector most faithfully. When m > n is the situation of interest, the coding is not unique, so we want the code to bear some structure, or in the Bayesian language, we want to impose some prior on the code. One possible structure/prior is reflected by the L_p-norm, i.e., we want to solve the following problem:

\[
\{\{z_i^*\}, D^*\} = \arg\min \sum_{i=1}^N \| x_i - D z_i \|_2^2 \\
\text{subject to } \| z_i \|_p \leq c
\]  

(1)

where D \in \mathbb{R}^{n \times m} is a linear decoding matrix, and both p and c are constants. In real data applications, we have difficulty in pre-determining the p value and instead the flexibility to learn the prior from the data becomes more important, which can be formulated as the following problem:

\[
\{\{z_i^*\}, D^*, p^*\} = \arg\min \sum_{i=1}^N \| x_i - D z_i \|_2^2 \\
\text{subject to } p \in \mathcal{P}, \forall i, \| z_i \|_p \leq c
\]  

(2)

where \mathcal{P} \subset \mathbb{R} is a domain of interesting p values. For example, \mathcal{P} = \{p | p \geq 1\} defines a domain that ensures the constraint to be convex with respect to z. In this paper, we term (1) and (2) L_p-norm constrained coding (p-ncc), and for distinguishing purpose we refer to (1) and (2) as p-ncc with known p and p-ncc with unknown p respectively.

B. Motivation

For p-ncc with known p, if the decoding matrix is given a priori as D^* = D_0, then it is sufficient to encode each x_i individually, by means of solving:

\[
z_i^* = \arg\min \| x_i - D_0 z_i \|_2^2 \\
\text{subject to } \| z_i \|_p \leq c
\]  

(3)

or its equivalent, unconstrained form (given a properly chosen Lagrange multiplier \lambda):

\[
z_i^* = \arg\min \| x_i - D_0 z_i \|_2^2 + \lambda \| z_i \|_p
\]  

(4)

the latter is well known as an L_p-norm regularized least squares problem, which arises in many disciplines of statistics, signal processing, and machine learning. Several special p values, such as 0, 1, 2 and \infty, have been intensively studied. For example, when p = 0, the term \| z_i \|_0 measures the number of non-zero entries in z_i, thus minimizing the term induces the code that is as sparse as possible. While sparsity is indeed a goal in several situations, the L_0-norm minimization is intractable (NP-hard [1]). Researchers then proposed to adopt L_1-norm, which gives rise to a convex and easier problem, to approach L_0-norm. It was shown that, under some mild conditions, the resulting code of using L_1-norm coincides with that of using L_0-norm [2]. L_1-norm regularization was previously known in lasso regression in statistics [3], and in basis pursuit in signal processing [4], and due to its enforced sparsity, leads to the great success of compressive sensing [5]. Non-sparse code was also investigated. L_2-norm regularization was extensively adopted under the name of weight decay in machine learning [6]. L_\infty-norm was claimed to help spread information evenly among the entries of the resultant code,
that is known as democratic [7] or spread representations [8], benefiting vector quantization and other applications [9].

Compared to the above-mentioned special $p$ values, other general $p$ values were much less studied due to mathematical difficulty. Nonetheless, it was observed that general $p$ indeed could help in specific application domains, where using the special $p$ values is over-simplified. On the one hand, $L_p$-norms with $p \in \{0, 1\}$ all enforce sparsity to some extent, but their effects are clearly different. In compressive sensing, it was known that using $L_p$-norm with $0 < p < 1$ needs fewer measurements to reconstruct sparse signal than using $L_1$-norm; regarding computational complexity, solving $L_p$-norm regularization is more difficult than solving $L_1$-norm but still easier than solving $L_0$-norm [10]. Further studies found that the choice of $p$ crucially affected the quality of results and the noise robustness [11]. Xu et al. endorsed the adoption of $L_{1/2}$-norm regularization and proposed a so-called iterative half thresholding algorithm for it [12]. In image deconvolution, Krishnan and Fergus investigated $L_{1/2}$ and $L_{2/3}$ norms and claimed their advantages over $L_1$-norm [13]. On the other hand, $L_p$-norms with $p > 1$ lead to non-sparse code, but still different $p$‘s have distinct impact on the solution. For example, Kloft et al. promoted the usage of $L_p$-norm with $p > 1$ in the task of multiple kernel learning [14].

Now let us return to the $p$-ncc problem (1) where the decoding matrix $D$ is unknown. As discussed above, $L_0$-norm induces sparse representations, thus the special case of $p$-ncc with $p = 0$ (similarly $p = 1$) is to pursue a sparse coding of the given data. While sparse coding has great interpretability, probably has relation to visual cognition [15], [16], and was widely adopted in many applications [17]–[20], we are inspired by the studies showing that general $p$ performs better than special $p$, and cannot help asking: is general $p$-ncc able to outperform sparse coding on a specific dataset for a specific task? If yes, then which $p$ will be the best (the $p$-ncc with unknown $p$)? These questions seem not being investigated before, to the best of our knowledge. Especially, “non-sparse” coding, i.e. $p$-ncc with $p > 1$, is clearly different from sparse coding and is the major theme of our study.

C. Outline of Solution

Analytically solving the $p$-ncc problems is very difficult as they are not convex optimization. When designing numerical solutions, note that there are two (resp. three) groups of variables in (1) (resp. (2)), it is natural to perform alternate optimization over them iteratively. Previous methods for sparse coding mostly follow this approach, for example in [15] and in the well-known K-SVD method [21]. One clear drawback of this approach is the high computational complexity due to the iterative nature. Even despite the drawback, it is still difficult to extend the sparse coding methods to the cases of general $p$ values, since they usually leverage the premise of sparse code heuristically but for general $p > 1$ it is hard to design such heuristics.

A different approach for sparse coding was proposed by Gregor and LeCun [22], where an iterative algorithm known as iterative shrinkage and thresholding (ISTA), that was previously used for $L_1$-norm regularized least squares, is unrolled and truncated to construct a multi-layer feed-forward network. The network can be trained end-to-end to act as a regressor from a vector to its corresponding sparse code. Note that truncation helps lower the computational cost of the network than the original algorithm, while training helps compensate the error due to truncation. The trained network known as learned ISTA (LISTA) is then a fast alternative to the original ISTA algorithm. Following the approach of LISTA, a few recent works consider the cases of $L_0$-norm [23] and $L_\infty$-norm [24], as well as extend other iterative algorithms to their network versions [25]–[27]. However, LISTA and its following-up works are not applicable for solving the $p$-ncc with general $p$, because they all refer to the same category of first-order iterative algorithms, i.e. the projected gradient descent (PGD) algorithms. For general $p$, the projection step in PGD is not analytically solvable. In addition, such works have more difficulty in solving the $p$-ncc with unknown $p$ problem.

Compared to the ISTA algorithm, the Frank-Wolfe algorithm is projection-free, which is also first-order iterative algorithm, but not belongs to the family of PGD. By avoiding projection, we show that the Frank-Wolfe algorithm can be adopted to solve the $p$-ncc with general $p$. In this paper, we unroll and truncate the Frank-Wolfe algorithm to construct a network, termed Frank-Wolfe network (F-W Net), which can be trained end-to-end. As previously known, the convergence speed of the Frank-Wolfe algorithm is very slow. We also show that F-W Net converges faster than not only the original Frank-Wolfe and ISTA algorithms, but also the LISTA. Moreover, F-W Net has a novel, closed-form and nonlinear computation unit that is parameterized by $p$, and as $p$ varies, that unit displays the behaviors of several classic pooling operators, and can be naturally viewed as a cascade of a generalized normalization and a parametric activation. Due to the fact that $p$ becomes a parameter in F-W Net, we can either set $p$ a priori or make $p$ learnable when training F-W Net. Thus, F-W Net has higher learning flexibility and training F-W Net gracefully solves the $p$-ncc with unknown $p$.

D. Our Contributions

To the best of our knowledge, we are the first to study $p$-ncc with general known $p > 1$ and $p$-ncc with unknown $p$; we are also the first to unroll-and-truncate the Frank-Wolfe algorithm to construct trainable network. Technically, we make the following contributions:

- We propose the Frank-Wolfe network (F-W Net), whose architecture is inspired by unrolling and truncating the Frank-Wolfe algorithm for solving $L_p$-norm regularized least squares problem. F-W Net features a novel nonlinear unit that is parameterized by $p$ and termed pool$_p$. The pool$_p$ unit links the conventional pooling, activation, and normalization operations in deep networks. F-W Net is verified to solve $p$-ncc with general known $p > 1$ better than the existing iterative algorithms. More importantly, F-W Net solves the $p$-ncc with unknown $p$ at low computational cost.

- We propose a convolutional version of F-W Net, which extends the basic F-W Net by adding convolutions and
utilizing the pool$_p$ unit pixel by pixel across different channels. The convolutional (Conv) F-W Net can be readily applied into image-related tasks.

- We evaluate the performance of F-W Net on an extensive range of simulations as well as the task of handwritten digit recognition, where F-W Net exhibits strong learning capability. We further apply Conv F-W Net into image denoising and super-resolution tasks, where F-W Net all demonstrates impressive effectiveness, flexibility, and robustness.

E. Paper Organization

The remainder of this paper is organized as follows. Section II reviews literatures from four-folds: sparse coding and its applications, deep structured networks, the original Frank-Wolfe algorithm, and nonlinear units in networks, from which we can see that F-W Net connects and integrates these separate fields. Section III formulates F-W Net, with detailed discussions on its motivation, structure, interpretation, and implementation issues. Section IV validates the performance of F-W Net on synthetic data under an extensive range of settings, as well as on a toy but real-world problem, i.e. handwritten digit recognition with the MNIST dataset. Section V discusses the proposed convolutional version of F-W Net and its potential applications. Section VI then provides experimental results of using Conv F-W Net on image denoising and super-resolution, with comparison to the state-of-the-art CNN models. Section VII concludes this paper. For reproducible research, our code and pre-trained models have been published online\(^1\).

II. RELATED WORK

A. Sparse Coding and Its Applications

Sparse coding as a representative methodology of the linear representation methods, has used widely in signal processing and computer vision, such as image denoising, deblurring, image restoration, super-resolution and image classification [17], [18], [20], [28]. The sparse representative aims to preserve the principle component and reduces the redundancies in the original signal. From the viewpoint of different norm minimizations used in sparsity constraints, these methods can be roughly categorized into the following groups: 1) $L_0$-norm minimization; 2) $L_p$-norm ($0 < p < 1$) minimization; 3) $L_1$-norm minimization; 4) $L_{2,1}$-norm minimization; and 5) $L_2$-norm minimization.

Varieties of dictionary learning methods have been proposed and implemented based on sparse representation. K-SVD [21] seeks an over-complete dictionary from given training signals under the $L_0$-norm constraint, which achieves good performance in image denoising. Wright et al. [28] proposes a general classification algorithm for face recognition based on $L_1$-norm minimization, which shows if the sparsity can be introduced into the recognition problem properly, the choice of features is not crucial. Krogh et al. [6] shows that limiting the growth of weights through $L_2$-Norm penalty can improve generalization in a feed-forward neural network. The simple weight decay has been adopted widely in machine learning.

B. Deep Structured Networks

Deep networks are typically stacked with off-the-shelf building blocks, that are jointly trained with simple loss functions. Since many real-world problems involve predicting statistically dependent variables or related tasks, deep structured networks [29], [30] were proposed to model complex patterns by taking into account such dependencies. Among many efforts, a conceivable portion has been devoted to unrolling the traditional optimization and inference algorithms into their deep feed-forward formats, that become end-to-end trainable.

Gregor and LeCun [22] first leveraged the idea to construct feed-forward networks as fast trainable regressors to approximate the sparse code solutions, whose idea was expanded by many successors, e.g. [23], [24], [31]–[34]. Those works show the benefits of incorporating the problem structure into the design of deep architectures, in terms of both performance and interpretability [35]. In [36], it was demonstrated theoretically that a deep network recovers $L_0$-based sparse representations under mild conditions. Note that previous works [22], [23], [31], [36] built their deep architectures based on the projected gradient descent-type algorithm, i.e., the iterative shrinkage and thresholding algorithm (ISTA). The projection step turned into the neuron function. Wang et al. [26] converted proximal methods to deep networks with continuous output variables, while the proximal operation also defined the neuron. More examples include the message-passing inference machine [25], shrinkage fields [37], CRF-RNN [38], and ADMM-net [27].

C. Frank-Wolfe Algorithm

The Frank-Wolfe algorithm [39], also known as conditional gradient descent, is one of the simplest and earliest known iterative solver, for the generic constrained convex problem:

$$
\min_z f(z) \quad \text{s.t. } z \in Z
$$

where $f$ is a convex and continuously differentiable objective function, and $Z$ is a convex and compact subset of a Hilbert space. At each step, the Frank-Wolfe algorithm first considers the linearization of $f(z)$, and then moves towards this linear minimizer that is taken over $Z$. Section III presents a concrete example of applying the Frank-Wolfe algorithm.

The Frank-Wolfe algorithm has lately re-gained popularity due to its promising applicability in handling structural constraints, such as sparsity or low rank. The Frank-Wolfe algorithm is projection-free: while competing methods such as the projected gradient descent and proximal algorithms need to take a projection step back to the feasible set per iteration, the Frank-Wolfe algorithm only solves a linear problem over the same set in each iteration, and automatically stays in the feasible set. For example, the sparsity-regularized problems are commonly relaxed as convex optimization over convex hulls of atomic sets, especially $\ell_p$ norm-constrained domains [40], which makes the Frank-Wolfe algorithm easily applicable. We refer the readers to the comprehensive review in [41], [42] for more details about the algorithm.

\(^1\)https://github.com/sunke123/FW-Net
D. Nonlinear Units in Networks

There have been blooming interests in designing novel neuron functions [43], a few of which have parametric and learnable forms, such as the parametric ReLU [44]. Among existing deep structured networks, e.g., [22], [23], [31], their neuron functions usually took fixed forms (e.g., some variants of ReLU) that reflected the pre-chosen structural priors. [45] presented a data-driven scheme to learn optimal thresholding functions for ISTA. Their adopted parametric representations led to spline curve-type neurons, which reduced the estimation errors compared to using the common (fixed) piece-wise linear ones.

As another major type of nonlinearity in deep networks, pooling was originally introduced as a dimension-reduction tool to aggregate a collection of inputs into low-dimensional outputs [20], [46]. Other than the input-output dimensions, the difference between neuron and pooling also lies in that neuron is typically applied element-wise, while pooling is on groups of hidden units, usually within a spatial neighborhood. [47] learned task-dependent pooling schemes and adaptively reshaped the pooling regions. [48] investigated more learnable pooling strategies, via either mixing two different pooling types or a tree-structured fusion. [49] introduced the \( L_p \) unit that computed a normalized \( \ell_p \) norm over the set of outputs, with the value of \( p \) learnable.

Finally, we find our proposed nonlinear unit inherently related to the normalization techniques. [46] demonstrated that a combination of neurons, pooling and normalization improved object recognition. Batch normalization (BN) [50] rescaled the summed inputs of neurons over training batches, and substantially accelerated training. Layer normalization (LN) [51] normalized the activations across all activities within a layer. Very recently, [52] re-exploited the idea of divisive normalization (DN) [53], [54], a well-grounded transformation in real neural systems. The authors viewed both BN and LN as the special cases of DN, and observed improvements by applying DN on a variety of tasks.

III. FRANK-WOLFE NETWORK

A. Frank-Wolfe Solver for \( L_p \)-Norm Constrained Least Squares

We investigate the \( \ell_p \)-Norm Constrained Least Squares (\( p \)-NCLS) problem as a concrete example in this paper, to illustrate the construction of F-W Net. We emphasize that the proposed methodology can certainly be extended to more generic problems (5). Let \( x \in \mathbb{R}^n \) denote the input data, and \( z \in \mathbb{R}^m \) denote the feature vector. \( D \in \mathbb{R}^{n \times m} \) denotes the bases (a.k.a. dictionary), with \( ||D||_2 = 1 \) by default to avoid the scale ambiguity of \( z \). \( p \)-NCLS considers \( Z \) to be an \( \ell_p \)-norm ball of radius \( c \), with \( p \geq 1 \) to ensure the convexity of \( Z \), and \( f(z) \) to be a least-squares loss:

\[
Z^* = \arg \min_z \frac{1}{2} ||x - Dz||_2^2 \quad \text{s.t.} \quad ||z||_p \leq c. \quad (6)
\]

We initialize \( z^0 \in Z \). At iteration \( t = 0, 1, 2, \ldots \), the Frank-Wolfe algorithm iteratively updates two variables \( z^t, s^t \in \mathbb{R}^m \) to solve (6):

\[
s^t := \arg \min_{s \in Z} s^T \nabla f(z^t) \quad (7)
\]

\[
z^{t+1} = (1 - \gamma^t)z^t + \gamma^t s^t, \quad (8)
\]

where \( \nabla f(z^t) = D^T (Dz^t - x) \), \( \gamma^t \) is the step size, which is typically set as \( \gamma^t := \frac{1}{f^2} \) or chosen via line search. By Hölder’s inequality, the solution of \( s^t \) is:

\[
s_i^t = -c \cdot \text{sgn}(\nabla_i f(z^t)) \frac{|\nabla_i f(z^t)|^{-1/2}}{(\sum_{j=1}^m |\nabla_j f(z^t)|^{p/2})^{1/2}}, \quad (9)
\]

where \( s_i^t, \nabla_i f(z^t) \) denote the \( i \)-th entry of \( s^t, \nabla f(z^t) \), respectively.

It is interesting to examine a few special \( p \) values in (9) (ignoring the negative sign for simplicity):

- \( p = 1, s^t \) selects the largest entry in \( \nabla f(z^t) \), while setting all other entries to zero\(^2\).
- \( p = 2, s^t \) re-scales \( \nabla f(z^t) \) by its root mean square.
- \( p = \infty, \) all entries of \( s^t \) have the equal magnitude \( c \).

Although the input and output are both \( \mathbb{R}^m \) and no dimensionality reduction is performed, the three special cases easily remind the behaviors of max pooling, root-mean-square pooling and average pooling [20], [46]. For general \( p \in [1, \infty) \), it is expected to exhibit more varied behaviors that can be interpretable from a pooling viewpoint. We thus denote by the function \( \mathbb{R}^m \rightarrow \mathbb{R}^m: s^t = \text{pool}_p(\nabla f(z^t)) \), to illustrate the operation by (9) associated with a specific \( p \).

B. Constructing the Network

Following the well-received “unrolling-then-truncating” methodology, e.g., [22], we represent the Frank-Wolfe solver (6), that runs finite \( T \) iterations \((t = 0, 1, \ldots, T - 1)\), as a multi-layer feed-forward neural network. Figure 1 depicts the resulting architecture, named the Frank-Wolfe Network (F-W Net). As a custom, we set \( z^0 = 0 \), which is an interior point of \( Z \) for any \( p \geq 1 \). The \( T \) layer-wise weights \( W_t \) are analytically constructed from \( D \):

\[
W_0 = -D^T; W_t = D^T D, t = 1, 2, \ldots, T - 1. \quad (10)
\]

Given the (pre-chosen) hyper-parameters, and without any further tuning, F-W Net outputs a \( T \)-iteration approximation \( z^T \) of the exact solution \( z \) to (6). As marked out, the intermediate outputs of F-W Net are aligned with the iterative variables in the original Frank-Wolfe solver (8). Note that the two-variable update scheme in (8) naturally leads to two groups of “skip connections”, which might be reminiscent of the deep residual network [56].

As many existing works did [22], [23], [31], the unrolled and truncated network in Figure 1 could be alternatively treated as a trainable regressor to predict the exact \( z \) from

\(^2\)This reminds us of the well-known Matching Pursuit algorithm. We noted that a recent work [55] has revealed a unified view between the Frank-Wolfe algorithm and the Matching Pursuit algorithm.
x. We further view F-W Net as composed from two sub-networks: (1) a single-layer initialization sub-network, consisting of a linear layer $W_0$ and a subsequent $pool_p$ operator. It provides a rough estimate of the solution: $z_1 = \gamma^s s_0 = \gamma^s pool_p(W_0 x)$, which appears similar to typical sparse coding that often initializes $z$ with a thresholded $D^T x$ [57]. Note that $z_1$ has a direct shortcut path to the output, with the multiplicative weights $\prod_{t=1}^{T-1} (1 - \gamma^t)$; (2) a recurrent refinement sub-network, that is unrolled to repeat the layer-wise transform for $T - 1$ times to gradually refine the solution.

F-W Net is designed to possess an ambitiously large learning flexibility, by fitting almost all its parameters and hyper-parameters from training data:

- **Layer-wise weights** $W_t$ are untied with $D$ and viewed as conventional fully-connected layers. All weights are learned with back-propagation from end to end.
- **Hyper-parameters** $p$ and $\gamma^t$ were given or pre-chosen in (8). In F-W Net, they can be either pre-chosen or made learnable. For learnable hyper-parameters, we also compute the gradients w.r.t. $p$ and $\gamma^t$, and update them via back-propagation. We adopt the same $p$ throughout F-W Net as $p$ implies the structural prior. $\gamma^t$ is set to be independent per iteration (layer). Learning $p$ and $\gamma^t$ also adaptively compensates for the truncation effects [45] of iterative algorithms.

F-W Net can further be jointly tuned with a task-specific loss function, e.g., the softmax loss for classification, in the form of an end-to-end network.

**C. Implementing the Network**

Reformulating $pool_p$ as normalization + neuron A closer inspection on the structure of the $pool_p$ function (9) leads to a further two-step decomposition (let $u^t = \nabla f(z^t)$) for simplicity:

$$
\begin{align*}
\gamma^t &= \frac{u^t}{||u^t||_p^{\frac{1}{p}}} \quad \text{ // Step 1: p-conjugate normalization} \\
\kappa^t_i &= -c \cdot \text{sgn}(y_i^t) \cdot |y_i^t|^{\frac{1}{p-1}} \quad \text{ // Step 2: p-exponential neuron} \\
\end{align*}
$$

Step 1 performs a normalization step under the $\frac{p}{p-1}$-norm. Let $q = \frac{p}{p-1}$, which happens to be the H"{o}lder conjugate of $p$; we thus call this step $p$-conjugate normalization. It coincides with a simplified form of DN [52], by setting all summation and suppression weights of DN to 1.

Step 2 takes the form as an exponential-type and non-saturated element-wise neuron [58], and is a learnable activation parameterized by $p$ [43]. As the output range of Step 1 is $[-1,1]$, the exponent displays suppression effects when $p \in (1,2)$, and amplifies entries when $p \in (2,\infty)$.

While the decomposition of (9) is surely not unique, we carefully choose (11) due to its effectiveness and numerical stability. As a counterexample, if we adopt another more straightforward decomposition of (9):

$$
y_i^t = \text{sgn}(u_i^t) \cdot |u_i^t|^{\frac{1}{p-1}}; \quad s_i^t = -c \cdot |y_i^t|^{\frac{1}{p}}
$$

then, large $|u_i^t|$ values (> 1) will be boosted by the power of $\frac{1}{p-1}$ when $p \in (1,2)$, and the second step may run the risk of “explosion” when $p \rightarrow 1$, in both feed-forward and back-propagation. In contrast, (11) first squashes each entry into $[-1,1]$ (Step 1) before feeding into the exponential neuron (Step 2), resolving the numerical issue well in practice.

The observation (11), called “$pool_p = \text{normalization} + \text{neuron}$” for brevity, provides a new interesting insight into the connections between neuron, pooling and normalization, three major types of nonlinear units in deep networks that were once considered separately. By splitting $pool_p$ into two modules sharing the parameter $p$, the back-propagation computation is also greatly simplified, as directly computing the gradient of $pool_p$ w.r.t. $p$ can be quite involved, with more potential numerical problems.

**Network Initialization and Training**

The training of F-W Net evidently benefits from high-quality initializations. Although $W_t$ and $D$ are disentangled, $W_0$ can be well initialized from the given or estimated $D^3$ via (10). $p$ is typically initialized with a large scalar, but its learning process is found to be quite insensitive to initialization. As observed in our experiments, no matter what $p$ is initialized as, it will converge

---

**Fig. 1.** (Please view in color) $T$-layer Frank-Wolfe network, consisting of two parts: (1) a single-layer initialization sub-network; and (2) a recurrent refinement sub-network, unrolled to $T - 1$ iterations. We use the *Green* notations to remind which variable in the original Frank-Wolfe solver (8) the current intermediate output is corresponding to. The layer-wise weights are denoted in *Black*, and the learnable hyper-parameters/units are in *Blue*. 

---

3When $D$ is not available, we estimate $D$ using KSVD [21] to initialize $W$ by default.
stably and smoothly to almost the same value\(^4\), \(\gamma^t\) is initialized with the rule \(\frac{2}{t+2}\), and is re-parametrized using a sigmoid function to enforce \([0, 1]\) range during training. \(c\) is the (only) hyper-parameter that needs to be manually chosen and fed into F-W Net. In experiments, we find that a good \(c\) choice could accelerate the convergence.

We have not seen a similar initialization strategy straightforwardly available to convolutional F-W Nets, as they are not directly tied with any dictionary. In all convolutional experiments, we thus use random initialization for the convolutional filters, while initializing \(p\) and \(\gamma^t\) in the same way as above. As a result, convolutional F-W Nets often converge slower than full-connected versions. We notice recent works as above. As a result, convolutional F-W Nets often converge slower than full-connected versions. We notice recent works

In the original \(p\)-NCLS, \(|\|D\|_2 = 1\) is assumed to avoid the scale ambiguity, and is commonly enforced in dictionary learning [21]. Similarly, to restrain the magnitude growth of \(W\), we adopt the \(\ell_2\)-norm weight decay regularizer, with the default coefficient \(2 \times 10^{-4}\).

D. Interpretation of Frank-Wolfe Network as LSTM

[35] interpreted the ISTA algorithm as a stack of plain Recurrent Neural Networks (RNNs). We hereby show that F-W Net can be potentially viewed as a special case of the Long Short-Term Memory (LSTM) architecture [61], which incorporates a gating mechanism [62] and may thus capture long-term dependencies better than plain RNNs when \(T\) is large. Although we include no such experiment in this paper, we are very much interested in applying F-W Net as LSTM and plan to exploit them as future options to initialize convolutional F-W Nets.

In the original \(p\)-NCLS, \(|\|D\|_2 = 1\) is assumed to avoid the scale ambiguity, and is commonly enforced in dictionary learning [21]. Similarly, to restrain the magnitude growth of \(W\), we adopt the \(\ell_2\)-norm weight decay regularizer, with the default coefficient \(2 \times 10^{-4}\).

IV. Evaluation of Frank-Wolfe Network

A. Simulations

We generate the synthetic data in the following steps. First we generate a random vector \(y \in \mathbb{R}^m\) and then we project it to the \(\ell_p\)-ball of radius \(c\). The projection is achieved by solving the problem: \(z = \arg \min_z \frac{1}{2}\|y - z\|_2^2 \text{s.t. } \|z\|_p \leq c\), using the original F-W algorithm until convergence. This \(p\) will be termed “ground-truth” \(p\) in the following. We then create a Gaussian random matrix \(D \in \mathbb{R}^{n \times m}\), and achieve \(x := Dz + e\), where \(e \in \mathbb{R}^n\) is additive white Gaussian noise with variance \(\sigma^2 = 0.01\). We use the default values \(m = 100\), \(n = 50\), \(c = 5\), and generate 15,000 samples for training. A testing set of 1,000 samples are generated separately for evaluation.

We first choose ground-truth \(p = 1.3\), vary \(T = 2, 4, 6, 8\), and compare the following methods:

- CVX: solving \(p\)-NCLS, using the ground-truth \(D\) and \(p = 1.3\). The CVX package [64] is employed to solve this convex problem. No training process is involved here.
- Original F-W: the baseline of running the original Frank-Wolfe algorithm for \(T\) iterations, using the ground-truth \(D\) and \(p = 1.3\) and fixing \(\gamma^t = \frac{2}{t+2}\). No training process is involved here.
- MLP: replacing the \(\text{pool}_p\) in F-W Net with ReLU, which is a plain feed-forward network of \(T\) fully-connected layers, having the same amount of parameters with F-W Net (except the prior \(p\)).
- F-W Net: the proposed network that jointly learns \(W_1\), \(p\), and \(\gamma^t\), \(t = 0, 1, \ldots, T - 1\).
- F-W fixed \{\(p, \gamma\)\}: fixing \(p = 1.3\) and \(\gamma^t = \frac{2}{t+2}\), learning \(W_1\) in F-W Net.
- F-W fixed \(\gamma\): fixing \(\gamma^t = \frac{2}{t+2}\), learning \(W_1\) and \(p\) in F-W Net.
- F-W fixed \{\(\text{wrong } p, \gamma\)\}: fixing \(p = 1\) (intentionally feeding an incorrect structural prior) and \(\gamma^t = \frac{2}{t+2}\), learning \(W_1\) in F-W Net.

Figure 2 (a) depicts the training curve of F-W net at \(T = 8\). We also run the Original F-W for comparison. Obviously after a few epochs, F-W Net is capable in achieving significantly smaller error than the Original F-W, and converges stably. It shows the advantage of training a network to compensate for the truncation error of limited iterations. It’s noteworthy that F-W Net has the ability to adjust the dictionary \(D\), prior \(p\) and step size \(\gamma^t\) at the same time, all of which are learned from training data. Comparing the test error of two methods at different \(T\)’s (Figure 2 (b)), F-W Net shows the superiority

\(^4\)We are aware of the option to re-parameterize \(p\) to ensure \(p \geq 1\) [49]. We have not implemented it in our experiments, since we never encountered \(p < 1\) during learning. It will be done in future.
of flexibility, especially in fewer steps or under the wrong condition. F-W Net learns to find a group parameters ($D$, $p$ and $\gamma^t$), so that these parameters coordinate each other leading to better performance, during the training process.

Figure 2 (b) compares the testing performance of different methods at different $T$ values. F-W Net with learnable parameters achieves the best performance. The Original F-W baseline performs poorly at $T = 2$, and its error barely decreases as $T$ increases to 4 and 8, since the original F-W algorithm is known to converge relatively slowly in practice. Note that CVX does not perform well because this problem is quite difficult ($p = 1.3$). Though having the same amount of parameters, MLP is unable to solve the p-NCLS well without the $L_p$-norm constraint, which indicates that the synthetic experiment is not a trivial linear regression problem. Especially, the original Frank Wolfe algorithm also significantly outperforms MLP after 4 steps.

Furthermore, we reveal the effect of each component ($D$, $p$ and $\gamma^t$). Firstly, we fix $p$ and $\gamma^t$ (shown as $F$-W fixed $\{p, \gamma\}$), where if $W_0 = -D^T$ and $W_1 = D^T D$, $t = 1, 2, ..., T - 1$, the solution of F-W Net is the same as the original F-W algorithm. By observing the two curves of testing performance, F-W Net obviously improve the performance with learnable $W_t$ which coordinates the fixed $p$ and $\gamma^t$ achieving better effect in approximating the target $z$ in fewer steps. Then, we let $p$ be learnable and only fix $\gamma^t$ (shown as $F$-W fixed $\gamma$), which is slightly superior to $F$-W fixed $\{p, \gamma\}$. F-W Net also maintains a smaller, but consistent margin over $F$-W fixed $\gamma$. Those three comparisons confirm that except for $W_t$, learning $p$ and $\gamma$ are both (crucially) useful. Finally, we give the wrong conditions (shown as $F$-W fixed $\{\text{wrong } p, \gamma\}$) to measure the influence on F-W Net. It is noteworthy that $F$-W fixed $\{\text{wrong } p, \gamma\}$ suffers from much larger error than other networks. That demonstrates the huge damage that an incorrect or inaccurate structural prior can cause to the learning process. But, as mentioned above, F-W Net has the high flexibility to adjust each component. Even though under the common $L_1$-norm constraint, F-W Net still outperforms the traditional algorithm in fewer steps.

Figure 3 inspects the learning of $p$ and $\gamma^t$. As seen from Figure 3 (a), the prior $p$ is initialized as 2 ($L_2$-norm), and the value fluctuates in the middle of training, but ends up converging stably. In Figure 3 (b), as $T$ goes up, the learned $p$ by F-W net approaches the ground-truth ($p = 1.3$) gradually. This phenomenon is consistent with that the original F-W algorithm cannot solve the problem in first steps with given $D$, $p$ and $\gamma^t$. Therefore, F-W Net adaptively controls each component through learning them from training data to approximate the distribution of $z$ as much as possible. To understand why the learned $p$ is usually larger than the ground-truth, one may intuitively think that pool$_p$ will “compress” the input more heavily as $p$ gets down closer to 1. To predict $z$, while the original Frank-Wolfe algorithm can run many iterations, F-W Net has to achieve the similar within a much smaller, fixed number of iterations. To this end, each pool$_p$ has to let more energy “pass through”, which we conjecture is responsible for the larger $p$ learned. Figure 3 (c) shows that the $p$ is learned by F-W Net with respect of different real $p$. When the “ground-truth” $p$ approaching 1 and 2, F-W Net is able to estimate the value of $p$ more accurately, as the convex problems with $L_1$-norm and $L_1$-norm minimization can be solved more easily. Figure 3 (d) observes the change of $\gamma^t$ before and after training, at $T = 8$. While $\gamma^0$ remains almost unchanged at 1, $\gamma^t$ ($t \geq 1$) all decreases. As a possible reason, F-W Net might try to compensate the truncation effects by raising the weight of the initialization $z^\dagger$ in the final output $z^T$.

We then look into $p = 1$ case, and re-generate data. We compare three baselines as similarly defined before: CVX, F-W Net, and F-W fixed $p$. In addition, we add LISTA [22] into comparison, which is equivalent to solving an unconstrained re-formulation of (6) when $p = 1$. The depth, layer-wise dimensions, and weight-sharing of LISTA are configured identically to F-W Net. Note that LISTA is dedicated to the $\ell_1$ case and cannot be easily adjusted for general $\ell_p$ cases. We re-tune all the parameters to get the best performance with LISTA to ensure a fair comparison.

Table 1 compares their results at $T = 2$, 4, and 6. The CVX is able to solve $\ell_1$ problems to a much better accuracy than the case of $p = 1.3$, and F-W Net still outperforms F-W fixed $p$. More interesting is the comparison between F-W Net and LISTA: F-W Net is outperformed by LISTA at $T = 2$, then reaches a draw at $T = 4$, and eventually outperforms LISTA by a large margin at $T = 6$. Increasing $T$ demonstrates a much more substantial impact on F-W Net than on LISTA, which can be interpreted as we have discussed in Section III that F-W Net is a special LSTM, and that LISTA can be interpreted as a vanilla RNN [35].
### B. Handwritten Digit Recognition

Similar to what had been done in [22], we adopt F-W Net as a feature extractor and then use logistic regression to classify the features. We design the following procedure to pre-train the F-W Net as a feature extractor. We use the MNIST dataset [65] to experiment. The original images are dimensionality reduced to 128-dim by PCA for input to F-W Net. Then we further perform PCA on each digit separately to reduce the dimension to 15. We construct a 150-dim sparse code for each image, whose 150 dimensions are divided into 10 groups to correspond to 10 digits, only 15 dimensions of which are non-zero and filled by the corresponding PCA result, whereas the other dimensions are all zero. This sparse code is regarded as the “ground-truth” for F-W Net in training. Accordingly, the transformation matrices in the second-step PCA are concatenated to serve as \( D \in \mathbb{R}^{128 \times 150} \), which is used to initialize the fully-connected layers in F-W Net according to (10). In this experiment, we use stochastic gradient descent (SGD) with weight decay of 0.0002, a momentum of 0.9 and a mini-batch size of 100. The F-W Net is pre-trained for 200 epochs, and the initial learning rate is 0.1, decayed to 0.01 at 100 epochs. Then the pre-trained F-W Net is augmented with a fully-connected layer (with softmax) that is randomly initialized, resulting a network that can be end-to-end trained for classification. We observe that the performance benefits from joint training marginally but consistently.

Due to the prior \( p \) unknown, most of current methods adopt an intuitive constraint, such as \( L_1 \)-norm and \( L_2 \)-norm. Different from them, F-W Net tries to attain a \( p \) from training data, which suits the real data better. The results are shown in Table II, comparing F-W Net with simple feed-forward fully-connected networks (MLP) [65] and LCoD [22]. F-W Net achieves lower error rate than the others. Especially, it takes 50 iterations for LCoD to achieve an error rate of 1.39, but only 3 equivalent layers for F-W Net to achieve 1.34, which has almost the same number of parameters as 50-iter LCoD. Moreover, with the increasing of layers, F-W Net makes a continuous improvement, which is consistent with the observation in the simulation.

Table III provides the results of different initializations of the hyper-parameter \( p \), showing that F-W Net is insensitive to the initialization of \( p \) and can converge to the learned \( p \approx 1.6 \). However, fixing the \( p \) value is not good for F-W Net even fixing to the finally learned \( p = 1.6 \). This is interesting as the learnable \( p \) provides advantages not only for the final trained model but also for training itself. An adjustable \( p \) value may suit for the evolving parameters during training F-W Net, which we plan to study further.

### V. Convolutional Frank-Wolfe Network

#### A. Constructing the Network

Previous similar works [22], [23], [31] mostly result in fully connected networks, as they are unrolled and truncated from linear sparse coding models. Nonetheless, fully connected networks are less effective than convolutional neural networks (CNNs) when tackling structured multi-dimensional signals such as images. A nature idea to extend this type of works to convolutional cases seems to turn to convolutional sparse coding [66], which also admits iterative solvers. However, the re-formulation will be inefficient both memory-wise and computation-wise, as discussed in [67].

We therefore seek a simpler procedure to build the convolutional F-W Net: in Figure 1, we replace the fully-connected layers with the convolutional layers, and operate \( pool_p \) across all the output feature maps for each location individually. The latter is inspired by the well-known conversion between convolutional and fully-connected layers by reshaping inputs\(^5\), and turns \( pool_p \) into a form of cross-channel parametric pooling [68]–[70].

The convolutional F-W Net then bears the similar flexibility to jointly learn weights and structural hyperparameters. Yet different from the original version of F-W Net, the \( pool_p \) in convolutional F-W Net reflects a diversified treatment of

\(^5\)http://cs231n.github.io/convolutional-networks/#convert

### Table I

| CVX       | F-W Net | F-W fixed p | LISTA   |
|-----------|---------|-------------|---------|
|           | \( T = 2 \) | \( T = 4 \) | \( T = 6 \) | \( T = 2 \) | \( T = 4 \) | \( T = 6 \) |
| MSE       | 0.0036  | 0.5641      | 0.3480  | 0.1961  | 1.2076 | 0.8604 | 0.7358 | 0.4157 | 0.3481 | 0.3053 |
| \( p \)   | 1 (fixed) | 1.360       | 1.254   | 1.222 | 1 (fixed) | 1 (fixed) | 1 (fixed) | 1 (fixed) | 1 (fixed) | 1 (fixed) |

### Table II

|          | params | error rate(%) |
|----------|--------|---------------|
| 3-layer-300+100 [65] | 286,200 | 3.05 |
| 3-layer-500+150 | 468,500 | 2.95 |
| 3-layer-500+300 | 545,000 | 1.53 |
| 1-iter LCoD [22] | 65,536 | 1.60 |
| 5-iter LCoD | 65,536 | 1.47 |
| 50-iter LCoD | 65,536 | 1.39 |
| 2-iter F-W Net | 63,350 | 1.20 |
| 3-iter F-W Net | 65,850 | 1.34 |
| 4-iter F-W Net | 88,200 | 1.25 |

### Table III

| Init \( p \) | Learnable \( p \) | Fixed \( p \) |
|-------------|-----------------|--------------|
| Accuracy (%) | \( p \) | Accuracy (%) | \( p \) |
| 1.1 | 98.51 | 1.601 | 97.35 | 1.1 |
| 1.3 | 98.45 | 1.600 | 97.70 | 1.3 |
| 1.5 | 98.43 | 1.595 | 98.30 | 1.5 |
| 1.6 | 98.66 | 1.614 | 98.26 | 1.6 |
| 1.8 | 98.38 | 1.601 | 97.90 | 1.8 |
| 2.0 | 98.50 | 1.585 | 97.78 | 2.0 |
different convolutional filters at the same local region, and should be differentiated from pooling over spatial locations.

- \( p = 1 \), only the channel containing the strongest response will be preserved at each spatial location, which is reduced to maxout [68].
- \( p = 2 \), \( \text{pool}_p \) re-scales the feature maps across the channels by its root mean square.
- \( p = \infty \), denotes the equal importance of all channels and leads to the cross-channel average “pooling”.

By involving \( p \) into optimization, \( \text{pool}_p \) essentially learns to re-scale local responses, as advocated by the neural lateral inhibition mechanism. The learned \( p \) will indicate the relative importance of different channels, and can potentially be useful for network compression [71].

In this paper, we extend convolutional F-W Net to low level image processing tasks, such as image denoising and image super-resolution. Our focus lies on building light-weight compact models and achieving superior performance through introducing the prior \( p \).

VI. EXPERIMENTS OF CONVOLUTIONAL FRANK-WOLFE NETWORK

A. Image Denoising

Despite decades of study, image denoising remains highly challenging. We investigate two versions of F-W Net for image denoising. The first version is inheriting the fully-connected F-W Net as used in the previous experiments. Here, the basic F-W Net (Figure 1) is augmented with one extra fully-connected layer, whose parameters are denoted by \( W_R \in \mathbb{R}^{n \times m} \) to reconstruct \( \hat{x} = W_Rz \). \( W_R \) is naturally initialized by \( D \). The network output is compared against the original (clean) image to calculate MSE loss. To utilize this fully-connected (FC) F-W Net, note that one strategy to image denoising is to split a noisy image into small (like \( 8 \times 8 \)) overlapping patches, process the patches, and compose the patches back to a denoised image. The classical KSVD+OMP algorithm follows this strategy, so we also follow it. Then, we re-shape \( 8 \times 8 \) blocks into 64-dimensional samples (i.e. \( n = 64 \)). We adopt \( m = 512 \), \( T = 8 \), and \( c = 1 \). The network is trained with a bunch of \( 8 \times 8 \) noisy blocks as well as their clean version. The second version is the proposed convolutional (Conv) F-W Net as discussed in Section V. The adopted configurations are: \( 3 \times 3 \) filters, 64 feature maps, \( T = 4 \), \( c = 1 \).

We use the BSD-500 dataset [72] to generate data. The BSD68 dataset is used as testing data, and the remaining 432 images are converted to grayscale and added with white Gaussian noise \( N(0, \sigma^2) \) for training. Four competitive baselines are chosen: a LISTA [22] network that is configured identically to F-W Net (FC); the classical KSVD+OMP [21] denoising algorithm and BM3D [73], using the dictionary size of 512. We would like to emphasize that the comparison with KSVD+OMP is unfair to us, since KSVD+OMP also leverages image self-similarity regularization by building the dictionary from the noisy image itself, which is not exploited by both LISTA and F-W Net. We also compare DnCNN [74] with (Conv) F-W Net, where we re-train DnCNN including 4 convolutions followed by BN+ReLU. To clearly demonstrate the flexibility and robustness of F-W Net, we adopt three standard noise levels: \( \sigma = 15 \), \( \sigma = 25 \) and \( \sigma = 50 \).

Table IV provides the results of different image denoising methods on the BSD 68 dataset. F-W Net(FC) is better than LISTA in all cases, which indicates the learnable prior \( p \) still superior to fixed \( p = 1 \). Furthermore, we also do ablation experiments to study the effect of \( p \), shown in Table V. F-W Net (learnable \( p \)) outperforms F-W Net (fixed \( p = 2 \)) by a big margin and is slightly better than F-W Net (fixed \( p = 1.4 \)). We emphasize that F-W Net has the ability to find a \( p \), which suits the given data. BM3D is a strong baseline, which the deep networks cannot beat very easily. As seen from Table IV, BM3D still outperforms DnCNN-T4, even though DnCNN-T20 can achieve better performance by introducing more parameters. F-W Net-T4 (Conv) has the advantage of exploiting the structure prior of data, which is help to recover the original signal from noise.

B. Image Super-Resolution

For the experiments on image super-resolution (SR), we compare FW-Net with baselines SRCNN [75] and VDSR [76]: all methods are trained on 91-image standard set and evaluated on Set-5/Set-14 benchmark with factor 3. We train a 4-layer F-W Net (Conv) and VDSR-T4 including 4 convolutional layer followed by ReLU, both of which has the same amount of parameters. We adopts the same training configurations as [76] and [75]. We prepare each training sample by randomly cropping 32x32-pixel sub-images from the training images, sub-sample it by the factor 3 and upscale it by the factor 3 via bicubic interpolation.

Table VI shows the comparisons of these three methods for image super-resolution. Following above, SR is also well-known to benefit from structural priors such as sparsity. F-W Net-T4 learns the prior \( p \) from the training data, and experimental results demonstrate its superiority. Moreover, we study the influence of each component on the final performance, provided in Table VII. We compare the following cases:

- F-W Net (No Skip): removing the top skip connections in Figure 1.

| TABLE IV | IMAGE DENOISING RESULTS (AVERAGE PSNR/DB) ON THE (GRAY) BSD-68 DATASETS. ALL THE NETWORKS INCLUDE 4 FULLY-CONNECTED OR CONVOLUTIONAL LAYERS. |
|----------|-----------------------|---------------------|---------------------|
|          | \( \sigma = 15 \)     | \( \sigma = 25 \)     | \( \sigma = 50 \)     |
| KSVD+OMP | 29.20                | 27.63               | 24.33               |
| LISTA-T4  | 29.35                | 27.71               | 24.52               |
| F-W Net-T4(FC), learned \( p = 1.41 \) | 31.07                | 28.57               | 25.62               |
| BM3D      | 30.89                | 28.42               | 25.42               |
| DnCNN-T4  | 31.27                | 28.71               | 25.66               |

| TABLE V | RESULTS OF F-W NET WITH FIXED \( p \) AND LEARNABLE \( p \) FOR IMAGE DENOISING ON THE BSD-68 DATASET. |
|----------|-----------------------|---------------------|---------------------|
|          | \( \sigma = 15 \)     | \( \sigma = 25 \)     | \( \sigma = 50 \)     |
| F-W Net(FC), fixed \( p = 2 \) | 28.35                | 26.41               | 23.68               |
| F-W Net(FC), fixed \( p = 1.4 \) | 29.25                | 27.62               | 24.45               |
| F-W Net(FC), learned \( p = 1.41 \) | 29.35                | 27.71               | 24.52               |


**Table VI**

| Method       | Set-5  | Set-14 |
|--------------|--------|--------|
| SRCNN        | 32.54  | 28.64  |
| VDSR-T4      | 32.51  | 28.71  |
| F-W Net-T4 (Conv) | 32.85  | 28.76  |

**Table VII**

| Method                        | Set-5  | Set-14 |
|-------------------------------|--------|--------|
| F-W Net-T4 (No Skip)          | 31.52  | -      |
| F-W Net-T4 (Fixed $\gamma = 1$) | 32.21  | -      |
| F-W Net-T4 (ReLU)             | 32.57  | 28.66  |
| F-W Net-T4                   | 32.85  | 28.76  |

- F-W Net-T4 (Fixed $\gamma = 1$): $\gamma^t = 1, t = 1,2,\ldots,T-1$ represents that the bottom skip connections are removed in Figure 1.
- F-W Net-T4 (ReLU): replacing all the pool$_p$ with ReLu, but keeping all the skip connections back.

F-W Net (No Skip) breaks the original optimization process and F-W Net-T4 (Fixed $\gamma = 1$) means that $z^{k+1}$ depends on the $s^k$ completely, both of which results in severe performance drop. F-W Net-T4 (ReLU) is similar to VDSR-T4, and also achieves the similar performance. These three comparisons are consistent to the observations in simulation experiments: each component in F-W Net contribute to the final performance.

Finally, we measure the effect of prior $p$, shown as the last rows in Table VII. The performance fluctuates widely with the change of the prior. F-W Net with learnable $p$ still achieves the best performance, which demonstrates the advantage of attaining the prior from the real data once again.

**VII. Conclusions**

The paper proposes the F-W Net, whose architecture is carefully designed by referring to a well-established optimization algorithm. Many aspects of F-W Net are inherently connected to the existing success of deep learning. It has gained impressive effectiveness, flexibility, and robustness in simulations and experiments. Results show that learning the hyper-parameter $p$ is quite beneficial especially in real-data experiments, which highlights the advantage of F-W Net in learning the $p$ during the end-to-end training.

A number of promising directions have emerged as our future work, including handling more structural constraints other than the $\ell_p$ ball, and the customization of F-W Net for more real-world applications.

**References**

[1] B. K. Natarajan, “Sparse approximate solutions to linear systems,” SIAM Journal on Computing, vol. 24, no. 2, pp. 227–234, 1995.

[2] D. L. Donoho, “For most large underdetermined systems of linear equations the minimal $\ell_1$-norm solution is also the sparest solution,” Communications on Pure and Applied Mathematics, vol. 59, no. 6, pp. 797–829, 2006.

[3] R. Tibshirani, “Regression shrinkage and selection via the lasso,” Journal of the Royal Statistical Society. Series B (Methodological), pp. 267–288, 1996.

[4] S. S. Chen, D. L. Donoho, and M. A. Saunders, “Atomic decomposition by basis pursuit,” SIAM Journal on Scientific Computing, vol. 20, no. 1, p. 33, 1998.

[5] D. L. Donoho, “Compressed sensing,” IEEE Transactions on Information Theory, vol. 52, no. 4, pp. 1289–1306, 2006.

[6] A. Krogh and J. A. Hertz, “A simple weight decay can improve generalization,” in Advances in neural information processing systems, 1992, pp. 950–957.

[7] C. Studer, T. Goldstein, W. Yin, and R. G. Baraniuk, “Democratic representations,” arXiv preprint arXiv:1401.3420, 2014.

[8] J.-J. Fuchs, “Spread representations,” in ASILOMAR, 2011, pp. 814–817.

[9] Y. Lyubarskii and R. Vershynin, “Uncertainty principles and vector quantization,” IEEE Transactions on Information Theory, vol. 56, no. 7, pp. 3491–3501, 2010.

[10] R. Chartrand, “Exact reconstruction of sparse signals via nonconvex minimization,” IEEE Signal Processing Letters, vol. 14, no. 10, pp. 707–710, 2007.

[11] R. Chartrand and V. Staneva, “Restricted isometry properties and nonconvex compressive sensing,” Inverse Problems, vol. 24, no. 3, p. 035020, 2008.

[12] Z. Xu, X. Chang, F. Xu, and H. Zhang, “$\ell_{1/2}$ regularization: A thresholding representation theory and a fast solver,” IEEE Transactions on Neural Networks and Learning Systems, vol. 23, no. 7, pp. 1013–1027, 2012.

[13] D. Krishnan and R. Fergus, “Fast image deconvolution using hyper-Laplacian priors,” in NIPS, 2009, pp. 1033–1041.

[14] M. Kloft, U. Brefeld, S. Sonnenburg, and A. Zien, “$\ell_p$-norm multiple kernel learning,” Journal of Machine Learning Research, vol. 12, no. Mar, pp. 953–997, 2011.

[15] B. A. Olshausen and D. J. Field, “Sparse coding with an overcomplete basis set: A strategy employed by V1?” Vision Research, vol. 37, no. 23, pp. 3311–3325, 1997.

[16] H. Lee, C. Ekanadham, and A. Y. Ng, “Sparse deep belief net model for visual area V2,” in NIPS, 2008, pp. 873–880.

[17] M. Elad and M. Aharon, “Image denoising via learned dictionaries and sparse representation,” in CVPR, vol. 1, 2006, pp. 895–900.

[18] J. Mairal, M. Elad, and G. Sapiro, “Sparse representation for color image restoration,” IEEE Transactions on Image Processing, vol. 17, no. 1, pp. 53–69, 2008.

[19] F. J. Huang, Y.-L. Boureau, Y. LeCun et al., “Unsupervised learning of invariant feature hierarchies with applications to object recognition,” in CVPR, 2007, pp. 1–8.

[20] J. Yang, K. Yu, Y. Gong, and T. Huang, “Linear spatial pyramid matching using sparse coding for image classification,” in CVPR, 2009, pp. 1794–1801.

[21] M. Aharon, M. Elad, and A. Bruckstein, “k-svd: An algorithm for designing overcomplete dictionaries for sparse representation,” IEEE Transactions on Signal Processing, vol. 54, no. 11, pp. 4311–4322, 2006.

[22] K. Gregor and Y. LeCun, “Learning fast approximations of sparse coding,” in ICML, 2010, pp. 399–406.

[23] Z. Wang, Q. Ling, and T. S. Huang, “Leaning deep $\ell_0$ encoders,” in AAAI, 2016, pp. 2194–2200.

[24] Z. Wang, Y. Yang, S. Chang, Q. Ling, and T. S. Huang, “Learning a deep $\ell_\infty$ encoder for hashing,” in IJCAI, 2016, pp. 2174–2180.

[25] S. Ross, D. Munoz, M. Hebert, and J. A. Bagnell, “Learning message-passing inference machines for structured prediction,” in CVPR, 2011, pp. 2737–2744.

[26] S. Wang, S. Fidler, and R. Urtasun, “Proximal deep structured models,” in NIPS, 2016, pp. 865–873.

[27] J. Sun, H. Li, Z. Xu et al., “Deep ADMM-net for compressive sensing MRI,” in NIPS, 2016, pp. 10–18.

[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, vol. 31, no. 2, pp. 210–227, 2009.

[29] L.-C. Chen, A. Schwing, A. Yuille, and R. Urtasun, “Learning deep structured models,” in International Conference on Machine Learning, 2015, pp. 1785–1794.

[30] A. G. Schwing and R. Urtasun, “Fully connected deep structured networks,” arXiv preprint arXiv:1503.02351, 2015.
