IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 4, 2021, accepted January 14, 2021, date of publication January 22, 2021, date of current version February 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3053764

Differentiable Forward and Backward Fixed-Point Iteration Layers

YOUNGHAN JEON[1], (Member, IEEE), MINSIK LEE[2], (Member, IEEE), AND JIN YOUNG CHOI[1], (Member, IEEE)

[1]Department of Electrical and Computer Engineering, ASRI, Seoul National University, Seoul 08826, South Korea
[2]Division of Electrical Engineering, Hanyang University, Ansan 15588, South Korea

Corresponding author: Jin Young Choi (jychoi@snu.ac.kr)

This work was supported in part by the Ministry of Science and Information and Communications Technology (ICT) Ministry of Science and ICT (MSIT), South Korea: {Outdoor Surveillance Robots} and { Information Technology Research Center (ITRC) through the Grand Information & Communication Technology Research Center support program}, supervised by the Institute for Information & Communications Technology Planning & Evaluation (IITP), under Grant IITP-2017-0-00306 and Grant IITP-2020-0-101741, and in part by the BK21.

ABSTRACT Recently, several studies have proposed methods to utilize some classes of optimization problems in designing deep neural networks to encode constraints that conventional layers cannot capture. However, these methods are still in their infancy and require special treatments, such as the analysis of the Karush–Kuhn–Tucker (KKT) condition, to derive the backpropagation formula. In this paper, we propose a new formulation called the fixed-point iteration (FPI) layer, which facilitates the use of more complicated operations in deep networks. The backward FPI layer, which is motivated by the recurrent backpropagation (RBP) algorithm, is also proposed. However, in contrast to RBP, the backward FPI layer yields the gradient using a small network module without explicitly calculating the Jacobian. In actual applications, both forward and backward FPI layers can be treated as nodes in the computational graphs. All the components of our method are implemented at a high level of abstraction, which allows efficient higher-order differentiations on the nodes. In addition, we present two practical methods, the neural net FPI (FPI_NN) layer and the gradient descent FPI (FPI_GD) layer, whereby the FPI update operations are a small neural network module and a single gradient descent step based on a learnable cost function, respectively. FPI_NN is intuitive and simple, while FPI_GD can be used to efficient train energy function networks that have been studied recently. While RBP and related studies have not been applied to practical examples, our experiments show that the FPI layer can be successfully applied to real-world problems such as image denoising, optical flow, and multi-label classification.

INDEX TERMS Fixed-point iteration, gradient descent, differentiable layers, recurrent back-propagation, energy network, deep learning architecture.

I. INTRODUCTION

Recently, several papers have proposed the composition of deep neural network with more complicated algorithms rather than the simple operations that have been used so far. For example, there have been methods using certain types of optimization problems in deep networks, such as differentiable optimization layers [4] and energy function networks [5], [9]. These methods can be used to introduce a prior into a deep network and provide the possibility of bridging the gap between deep learning and some of the traditional methods. However, they are still premature and require non-trivial efforts to implement in actual applications. In particular, the backpropagation formula has to be derived explicitly for each different formulation based on some criteria such as the KKT conditions. This limits the practicality of the approaches, since there can be numerous different formulations depending on the actual problems. As far as we know, this is the first attempt to address these issues. Unlike the aforementioned differentiable optimization layers, the proposed method can embed an optimization problem into a neural network without deriving a separate backpropagation

The associate editor coordinating the review of this manuscript and approving it for publication was Fanbiao Li.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

formula depending on the problem. Furthermore, energy function networks can be trained efficiently using our method.

Several decades ago, an algorithm called the recurrent backpropagation (RBP) was proposed by Almeida [3] and Pineda [22]. It is a method of training a recurrent neural network (RNN) that converges to the steady state. The advantages of RBP are that it can be applied universally to most operations that consist of repeated computations and that the whole process can be summarized in a single update equation. Even with its long history, however, RBP and related studies [6], [20] have been tested only for the verification of theoretical concepts, and there have been no examples applying these methods to practical tasks. Moreover, there have been no studies using RBP in conjunction with other neural network components to verify its effects in more complex settings. We add several improvements to the existing advantages of RBP in the backpropagation process. Our formula can be applied to general problems as well as combined with other network components. Unlike RBP-based studies, our method shows good performance on practical tasks.

To facilitate the use of more complicated operations in deep networks, in this paper, we introduce a new layer formulation that can be practically implemented and trained based on RBP with some additional considerations. To this end, we employ the fixed-point iteration (FPI), which is the basis for many numerical algorithms, including most gradient-based optimizations, as a *layer* in the neural network. The input and weights of the FPI layer are used to define an update equation, and the output of the layer is the fixed point of the update equation. Under mild conditions, the FPI layer is differentiable and the derivative depends only on the fixed point, which is much more efficient than adding all the individual iterations to the computational graph.

We also propose a backpropagation method called the *backward FPI layer* based on RBP [3], [22] to efficiently compute the derivative of the FPI layer. We prove that, if the aforementioned conditions for the FPI layer hold, then the backward FPI layer also converges. In contrast to RBP, the backward FPI layer yields the gradient through a small network module, which allows us to avoid explicitly calculating the Jacobian. In other words, we do not need a separate derivation for the backpropagation formula, and we can utilize existing autograd functionalities. Specifically, we provide a modularized implementation of the partial differentiation operation, which is essential in the backward FPI layer but absent from regular autograd libraries, based on an independent computational graph. This makes the proposed method very convenient for various practical applications. Since FPI covers many different types of numerical algorithms as well as optimization problems, there are numerous potential applications for the proposed method. The FPI layer is highly modularized, so it can be used easily with other existing layers, such as the convolution and rectified linear unit (ReLU),[1] and it has richer representation power than a feedforward layer with the same number of weights. The contributions of the paper are summarized as follows:

- We propose the use of FPI as a layer in the neural network. The FPI layer can incorporate the mechanisms of conventional iterative algorithms, such as numerical optimization, into deep networks. Unlike other existing layers based on differentiable optimization problems, this implementation is much simpler, and the backpropagation formula can be universally derived.
- For backpropagation, the backward FPI layer is proposed based on RBP to compute the gradient efficiently. Under mild conditions, we show that both forward and backward FPI layers are guaranteed to converge. All components are highly modularized, and a general partial differentiation tool is developed so that the FPI layer can be used in various circumstances without any modifications.
- Two types of FPI layers (FPI_NN and FPI_GD) are presented. Proposed networks based on the FPI layers are applied to practical tasks such as image denoising, optical flow, and multi-label classification. These tasks have been largely absent in existing RBP-based studies, and the networks show good performance.

The remainder of this paper is organized as follows: We first introduce related works in Section II. The proposed FPI layer is explained in Section III, and the experimental results follow in Section IV. Finally, we conclude the paper in Section V.

II. BACKGROUND AND RELATED WORK

Fixed-point iteration (FPI): For a given function g and a sequence of vectors $\{x_n \in \mathbb{R}^d\}$, the FPI [10] is defined by the following update equation:

$$x_{n+1} = g(x_n), \quad n = 0, 1, 2, \cdots, \tag{1}$$

which converges to a fixed point $\hat{x}$ of g, satisfying $\hat{x} = g(\hat{x})$. The gradient descent method ($x_{n+1} = x_n - \gamma \nabla f(x_n)$) is a popular example of FPI. Many numerical algorithms are based on FPI, and there are also many examples in machine learning. Some important concepts about FPI are listed below.

Definition 1 (Contraction Mapping) [17]: On a metric space (X, d), the function $f : X \to X$ is a contraction mapping if there is a real number $0 \leq k < 1$ that satisfies the following inequality for all x_1 and x_2 in X:

$$d\big(f(x_1), f(x_2)\big) \leq k \cdot d(x_1, x_2). \tag{2}$$

The smallest k that satisfies the above condition is called the Lipschitz constant of f. The distance metric is defined as an arbitrary norm $\|\cdot\|$ in this paper. Based on the above definition, the Banach fixed-point theorem [7] states the following:

[1] The code will be available upon publication.

Theorem 1 (Banach Fixed-Point Theorem): A contraction mapping has exactly one fixed point, and it can be found by starting with any initial point and iterating the update equation until convergence.

Therefore, if g is a contraction mapping, it converges to a unique point $\hat{x}$ regardless of the starting point x_0. The above concepts are important in deriving the proposed FPI layer in this paper.

Energy function networks: Scalar-valued networks that estimate energy (or error) functions have attracted considerable research interest recently. The energy function network (EFN) has a different structure from general feed-forward neural networks, and the concept was first proposed in [19]. After training an EFN for a certain task, the answer to a test sample is obtained by finding the input of the trained EFN that minimizes the network's output. The structured prediction energy network (SPEN) [9] performs a gradient descent on an EFN to find the solution, and a structured support vector machine [23] loss is applied to the obtained solution. Input convex neural networks (ICNNs) [5] are defined in a specific way such that the networks have convex structures with respect to (w.r.t.) the inputs, and their learning and inference are performed by the entropy method, which is derived based on the KKT optimality conditions. Deep value networks [14] and the IoU-Net [15] directly learn the loss metrics, such as the intersection over union (IoU) of bounding boxes, and then perform inference through gradient-based optimization methods.

Although the above approaches provide novel ways of utilizing neural networks in optimization frameworks, they have not been combined with other existing deep network components to verify their effectiveness in more complicated problems. Moreover, they are mostly limited to a certain type of problem and require complicated learning processes. Our method can be applied to broader situations than existing EFN approaches, and these approaches can be equivalently implemented by the proposed method once the update equation for the optimization problem is derived.

Differentiable optimization layers: Recently, some papers have proposed using optimization problems as a layer in the deep learning architecture. Such structures can exhibit more complicated behavior in one layer than the usual layers in neural networks, and they can potentially reduce the depth of the network. OptNet [4] presents how to use the quadratic program (QP) as a layer of a neural network. They also use the KKT conditions to compute the derivative of the solution of QP. Agrawal *et al.* [1] propose an approach to differentiate disciplined convex programs, which are subclass of convex optimization problems. Some other researches have tried to differentiate optimization problems such as submodular models [11], cone programs [2], and semidefinite programs [24]. However, most of these have limited applications, and users need to adapt their problems to the rigid problem settings. On the other hand, our method facilitates the use of a large class of iterative algorithms, which also includes differentiable optimization problems, as a network layer.

Recurrent backpropagation: RBP is a method to train a special case of RNNs proposed by Almeida [3] and Pineda [22]. It computes the gradient of the steady state for an RNN with constant memory. Although RBP has great potential, it is rarely used for practical problems in deep learning. Some artificial experiments have been performed to show its memory efficiency, but it has been difficult to apply in complex and practical tasks. Recently, Liao *et al.* [20] tried to revive RBP using the conjugate gradient method and the Neumann series. However, both the forward and backward passes use a fixed number of steps (maximum 100), which may not be sufficient for convergence in practical problems. In addition, if the forward pass does not converge, the equilibrium point would be meaningless; hence, it may be unstable to train the network using the unconverged final point, which is a problem that is not addressed in the paper. Deep equilibrium models (DEQs) [6] have tried to find the equilibrium points of a deep sequence model via an existing root-finding algorithm. For backpropagation, they then compute the gradient of the equilibrium point using another root-finding method. In short, both the forward and backward passes are implemented via quasi-Newton methods. DEQs can also be executed with constant memory, but they can only model sequential (temporal) data, and the aforementioned convergence issues still exist.

RBP-based methods mainly perform experiments to verify theoretical concepts and have not been widely applied to practical examples. Our work incorporates the concept of RBP into the FPI layer to apply complicated iterative operations in deep networks and presents two types of algorithms accordingly. The proposed method is the first RBP-based method that shows successful applications to practical tasks in machine learning or computer vision, and it can be widely used for promoting RBP-based research in the field of deep learning.

III. PROPOSED METHOD

The FPI formula contains a wide variety of forms and can be applied to most iterative algorithms. Section III-A describes the basic structure and principles of the FPI layer. Sections III-B and III-C explain the differentiation of the layer for backpropagation. Section III-D describes the convergence of the FPI layer, and Section III-E presents two example cases of the layer.

A. STRUCTURE OF THE FIXED-POINT ITERATION LAYER

Here, we describe the basic operation of the FPI layer. Let $g(x, z; \theta)$ be a parametric function where x and z are vectors of real numbers and θ is the parameter. We assume that g is differentiable for x and has a Lipschitz constant of less than one for x, and the following fixed point iteration converges to a unique point according to the Banach fixed-point iteration theorem:

$$x_{n+1} = g(x_n, z; \theta), \quad \hat{x} = \lim_{n \to \infty} x_n \tag{3}$$

The FPI layer, $\mathbf{F}$, can be defined based on the above relations. It receives an observed sample or output from the previous layer as input z and yields the fixed point $\hat{x}$ of g as its output; i.e.,

$$\begin{aligned}\hat{x} &= g(\hat{x}, z; \theta) = \mathbf{F}(x_0, z; \theta) \\ &= \lim_{n\to\infty} g^{(n)}(x_0, z; \theta) = (g \circ g \circ \cdots \circ g)(x_0, z; \theta), \quad (4)\end{aligned}$$

where $\circ$ indicates the function composition operator. The layer receives the initial point x_0 as well, but its actual value does not matter in the training procedure, because g has a unique fixed point. Hence, x_0 can be preset to any value, such as a zero matrix. Accordingly, we often express $\hat{x}$ as a function of z and θ in this paper; i.e., $\hat{x}(z; \theta)$. When using an FPI layer, the first equation in (3) is repeated until convergence to find the output $\hat{x}$. We may use some acceleration techniques such as Anderson acceleration [21] for faster convergence. Note that there is no apparent relation between the shapes of x_n and z. Hence, the sizes of the input and output of an FPI layer do not have to be equal.

B. DIFFERENTIATION OF THE FPI LAYER

Similar to other network layers, the learning of $\mathbf{F}$ is performed by updating θ based on backpropagation. To this end, the derivatives of the FPI layer have to be calculated. One simple way to compute the gradients is to construct a computational graph for all iterations up to the fixed point $\hat{x}$. However, this method is not only time consuming but also requires a large amount of memory.

In this section, we show that the derivative of the entire FPI layer depends only on the fixed point $\hat{x}$. In other words, all the x_n before the convergence are not actually needed to compute the derivatives. Hence, we can solely retain the value of $\hat{x}$ to perform backpropagation, and we consider the entire $\mathbf{F}$ as a node in the computational graph. Note that $\hat{x} = g(\hat{x}, z; \theta)$ is satisfied at the fixed point $\hat{x}$. If we differentiate both sides of the above equation w.r.t. θ, we have

$$\frac{\partial \hat{x}}{\partial \theta} = \frac{\partial g}{\partial \theta}(\hat{x}, z; \theta) + \frac{\partial g}{\partial x}(\hat{x}, z; \theta)\, \frac{\partial \hat{x}}{\partial \theta}. \quad (5)$$

Here, z is not differentiated, because z and θ are independent. Rearranging the above equation gives

$$\frac{\partial \hat{x}}{\partial \theta} = \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta)\right)^{-1} \frac{\partial g}{\partial \theta}(\hat{x}, z; \theta), \quad (6)$$

which confirms that the derivative of the output of $\mathbf{F}(x_0, z; \theta) = \hat{x}$ depends only on the value of $\hat{x}$. One downside to the above derivation is that it requires the calculation of the Jacobians of g, which may take much memory space (e.g., convolutional layers). Moreover, calculating the inverse can also be a burden. The next section provides an efficient way to resolve these issues.

C. BACKWARD FIXED-POINT ITERATION LAYER

To train the FPI layer, we need to obtain the gradient w.r.t. its parameter, θ. In contrast to RBP [3], [22], we propose a computationally efficient layer, called the backward FPI layer, which yields the gradient without explicitly calculating the Jacobian. Here, we assume that the FPI layer is in the middle of the network. If we define the loss of the entire network as L, then what we need for backpropagation of the FPI layer is $\nabla_\theta L(\hat{x})$. According to (6), we have

$$\begin{aligned}\nabla_\theta L &= \left(\frac{\partial \hat{x}}{\partial \theta}\right)^\top \nabla_{\hat{x}} L \\ &= \left(\frac{\partial g}{\partial \theta}(\hat{x}, z; \theta)\right)^\top \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta)\right)^{-\top} \nabla_{\hat{x}} L. \quad (7)\end{aligned}$$

This section describes how to calculate the above equation efficiently. (7) can be divided into two steps as follows:

$$c = \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta)\right)^{-\top} \nabla_{\hat{x}} L, \quad (8)$$

$$\nabla_\theta L = \left(\frac{\partial g}{\partial \theta}(\hat{x}, z; \theta)\right)^\top c. \quad (9)$$

Rearranging (8) yields $c = \left(\frac{\partial g}{\partial x}(\hat{x}, z; \theta)\right)^\top c + \nabla_{\hat{x}} L$, which can be expressed as an iteration form, i.e.,

$$c_{n+1} = \left(\frac{\partial g}{\partial x}(\hat{x}, z; \theta)\right)^\top c_n + \nabla_{\hat{x}} L, \quad (10)$$

which corresponds to RBP. This iteration eliminates the need for the inverse calculation, but it still requires the calculation of the Jacobian of g w.r.t. $\hat{x}$. Here, we derive a new network layer, the backward FPI layer, which yields the gradient without explicitly calculating the Jacobian. To this end, we define a new function, h, as $h(x, z, c; \theta) = c^\top g(x, z; \theta)$, and (10) becomes

$$c_{n+1} = \frac{\partial h}{\partial x}(\hat{x}, z, c_n; \theta) + \nabla_{\hat{x}} L. \quad (11)$$

Note that the output of h is scalar. Here, we can consider h as another small network containing only a single step of g (with an additional inner product). The gradient of h can be computed based on existing autograd functionalities with some additional considerations. Similarly, (9) is expressed using h as follows:

$$\nabla_\theta L = \frac{\partial h}{\partial \theta}(\hat{x}, z, c; \theta), \quad (12)$$

where c is the fixed point obtained from the FPI in (11). In this way, we can compute $\nabla_\theta L$ using (12) without any memory-intensive operations or Jacobian calculations. c can be obtained by initializing c to some arbitrary value and repeating the above update until convergence.

Note that this backward FPI layer can be treated as a node in the computational graph, hence the name "backward FPI *layer*". However, care should be taken regarding the above derivation, as the differentiations w.r.t. x and θ are *partial*

differentiations. x and θ may have some dependency on each other, which can disrupt the partial differentiation process if it is computed based on the usual autograd framework. Hereafter, let $\phi(a, b)$ denote the gradient operation in the conventional autograd framework that calculates the derivative of a w.r.t b, where a and b are both nodes in a computational graph. Here, b can also be a set of nodes, in which case the output of ϕ would also be a set of derivatives.

In order to resolve the issue, we implement a general partial differentiation operator $P(s; r) \triangleq \partial r(s)/\partial s$, where s is a set of nodes, r is a function (a function object, to be precise), and $\partial r(s)/\partial s$ denotes the set of corresponding partial derivatives. Let $I(t)$ denote an operator that creates a set of leaf nodes by cloning the nodes in the set t and detaching them from the computational graph. P first creates an independent computational graph having leaf nodes $s' = I(s)$. These leaf nodes are then passed onto r to yield $r' = r(s')$, and now we can differentiate r' w.r.t. s' using $\phi(r', s')$ to calculate the partial derivatives, because the nodes in s' are independent of each other. Here, the resulting derivatives $\partial r'/\partial s'$ are also attached to the independent graph as the output of ϕ. The P operator creates another set of leaf nodes $I(\partial r'/\partial s')$, which is then attached to the original graph (where s resides) as the output of P, i.e., $\partial r(s)/\partial s$. In this way, the whole process is completed and the partial differentiation can be performed accurately. If some of the partial derivatives are not needed in the process, we can simply omit them in the calculation of $\partial r'/\partial s'$.

Note that the above independent graph is preserved for backpropagation. Let $H(v; u)$ be an operator that creates a new function object that calculates $\sum_i \langle v_i, u_i \rangle$, where the node v_i is an element of the set v and u_i is one of the outputs of the function object u. In the backward path of P, the set δ of gradients passed onto P by backpropagation is used to create a function object $\eta = H(\delta; \rho)$ where $\rho(s)$ is a function object that calculates $P(s; r) = \partial r(s)/\partial s$. The backpropagated gradients for s can be calculated with another P operation; i.e., $P(\delta \cup s; \eta)$ (here, the derivatives for δ do not need to be calculated). In practice, the independent graph created in the forward path is reused for ρ in calculating η.

The backward FPI layer can be highly modularized with the above operators (i.e., P, H); a plus operator can construct (11) and (12) entirely, and the iteration of (11) can be implemented with another forward FPI layer. This allows multiple differentiations of the forward and backward FPI layers. A picture depicting all the above processes is provided in the supplementary materials. All the forward and backward layers are implemented at a high level of abstraction; therefore, it can be easily applied to practical tasks by changing the structure of g to one that is suitable for each task.

D. CONVERGENCE OF THE FPI LAYER

The forward path of the FPI layer converges if the bounded Lipschitz assumption holds. For example, to make a fully connected layer a contraction mapping, simply dividing the weight by a number greater than the maximum singular value of the weight matrix will suffice. Empirically, we found that setting the initial values of weights (θ) to small values is enough to make g a contraction mapping throughout the training procedure.

Convergence of the backward FPI layer. The backward FPI layer is composed of a linear mapping based on the Jacobian $\partial g/\partial x$ on $\hat{x}$. Convergence of the backward FPI layer can be confirmed by the following proposition:

Proposition 1: If g is a contraction mapping, the backward FPI layer (10) *converges to a unique point.*

Proof: For simplicity, we omit z and θ from g. By the definition of contraction mapping and the assumption of the arbitrary norm metric, $\frac{\|g(x_2)-g(x_1)\|}{\|x_2-x_1\|} \leq k$ is satisfied for all x_1 and x_2 ($0 \leq k < 1$) by inequality (2). For a unit vector v, i.e., $\|v\| = 1$ for the aforementioned norm, and a scalar t, let $x_2 = x_1 + tv$. The above inequality then becomes $\frac{\|g(x_1+tv)-g(x_1)\|}{\|t\|} \leq k$. For another vector u with $\|u\|_* \leq 1$ where $\|\cdot\|_*$ indicates the dual norm, it satisfies

$$\frac{u^\top\big(g(x_1+tv)-g(x_1)\big)}{|t|} \leq \frac{\|g(x_1+tv)-g(x_1))\|}{|t|} \leq k \tag{13}$$

based on the definition of the dual norm. This indicates that

$$\lim_{t\to 0^+} \frac{u^\top\big(g(x_1+tv)-g(x_1)\big)}{|t|} = \nabla_v(u^\top g)(x_1) \leq k. \tag{14}$$

According to the chain rule, $\nabla(u^\top g) = \left(u^\top J_g\right)^\top$ where J_g is the Jacobian of g. This yields

$$\begin{aligned} \nabla_v(u^\top g)(x_1) &= \big(\nabla(u^\top g)(x_1)\big)^\top \cdot v \\ &= u^\top J_g(x_1)\, v \leq k. \end{aligned} \tag{15}$$

Let $x_1 = \hat{x}$ then $u^\top J_g(\hat{x})\, v \leq k$ for all u, v that satisfy $\|u\|_* \leq 1$, $\|v\| = 1$. Therefore,

$$\begin{aligned} \|J_g(\hat{x})\| &= \sup_{\|v\|=1} \|J_g(\hat{x})v\| \\ &= \sup_{\|v\|=1, \|u\|_*\leq 1} u^\top J_g(\hat{x})v \leq k < 1, \end{aligned} \tag{16}$$

which indicates that the linear mapping by weight $J_g(\hat{x})$ is a contraction mapping. By the Banach fixed-point theorem, the backward FPI layer converges to the unique fixed point. □

E. TWO REPRESENTATIVE CASES OF THE FPI LAYER

As mentioned above, FPI can take a wide variety of forms. We present two representative methods that are easily applicable to practical problems.

1) NEURAL NET FPI (FPI_NN) LAYER

The most intuitive way to use the FPI layer is to set g as an arbitrary neural network module. In FPI_NN, the input variable recursively enters the same network module until convergence. g can be composed of layers that are widely used in deep networks, such as convolution, ReLU, and linear layers. The FPI_NN approach can perform more complicated behaviors with the same number of parameters than using g directly without FPI, as demonstrated in the experiments section.

2) GRADIENT DESCENT FPI (FPI_GD) LAYER

The gradient descent method can be a perfect example for the FPI layer. It can be used for efficient implementations of the EFN, such as the ICNN [5]. Unlike a typical network, which obtains the answer directly as the output of the network (i.e., $f(a;\theta)$ is the answer to the query a), an EFN retrieves the answer by optimizing an input variable of the network (i.e., $\text{argmin}_x f(x, a; \theta)$ becomes the answer). The easiest way to optimize the network f is through gradient descent ($x_{n+1} = x_n - \gamma \nabla f(x_n, a; \theta)$). This is a form of FPI, and the fixed point $\hat{x}$ is the optimal point of f, i.e., $\hat{x} = \text{argmin}_x f(x, a; \theta)$. In the case of a network with a single FPI_GD layer, $\hat{x}$ becomes the final output of the network. Accordingly, this output is fed into the final loss function $L(x^*, \hat{x})$ to train the parameter θ during the training procedure. This behavior conforms to that of an EFN. However, unlike existing methods, the proposed method can be trained easily with the universal backpropagation formula. Therefore, the proposed FPI layer can be an effective alternative for training EFNs. One advantage of FPI_GD is that it can easily satisfy the bounded Lipschitz condition by adjusting the step size γ.

IV. EXPERIMENTS

Since several studies [3], [6], [20], [22] have already shown that RBP-based algorithms require only a constant amount of memory, we omitted memory-related experiments. Instead, we focused on applying the proposed method to practical tasks in this paper. It is worth noting that both the forward and backward FPI layers were highly modularized, and the exactly same implementations were shared across all the experiments without any alterations. The only difference was the choice of g, into which we could simply plug in its functional definition; this shows the efficiency of the proposed framework. In the image denoising experiment, we compared the performance of the FPI layer to a non-FPI network that has the same structure as g. In the optical flow problem, a relatively very small FPI layer is attached at the end of FlowNet [12] to demonstrate its effectiveness. For all the experiments, the detailed structure of g and the hyper-parameters for training are described in the supplementary materials. Results for the multi-label classification problem show that the FPI layer is superior in performance compared to the existing state-of-the-art algorithms. All training was performed using the Adam [18] optimizer.

A. IMAGE DENOISING

Here, we compared the image denoising performance for gray images perturbed by Gaussian noise with variance σ^2. Traditionally, image denoising has been solved with iterative, numerical algorithms; hence, an iterative structure such as the proposed FPI layer can be an appropriate choice for the problem. To generate the image samples, we cropped the images in the Flying Chairs dataset [12] and converted them to grayscale (400 images for training and 100 images for testing). We constructed a single FPI_NN layer network for this experiment. For comparison, we also constructed a feedforward network that has same structure as g. The performance is reported in terms of peak signal-to-noise ratio (PSNR) in Table 1.

TABLE 1. Denoising performance (PSNR, higher is better). The single FPI_NN network outperformed the feedforward network, and the performance gap was larger for noisier (more difficult) circumstances..

Method	$\sigma = 15$	$\sigma = 20$	$\sigma = 25$
Feedforward	32.18	30.44	29.09
FPI_NN	**32.43**	**31.00**	**29.74**

Table 1 shows that the single FPI layer network outperformed the feedforward network across all experiments. Note that the performance gap between the two algorithms was larger under noisier circumstances. Since both networks were trained to yield the best performance with their given settings, this confirms that a structure with repeated operations can be more suitable for this type of problem. One advantage of the proposed FPI layer here is that there is no explicit calculation of the Jacobian, which can be quite large in this image-based problem, even though there is no specialized component except the bare definition of g, thanks to the highly modularized nature of the layer. Examples of the image denoising results are shown in the supplementary materials.

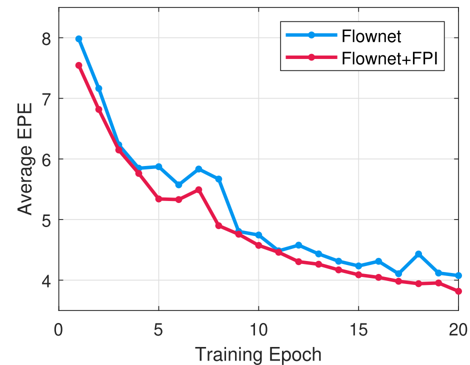

FIGURE 1. Average EPE per epoch (lower is better). A very small FPI layer helps to refine the FlowNet algorithm..

B. OPTICAL FLOW

Optical flow, one of the major research areas in computer vision, aims to acquire motions by matching pixels in two images. We demonstrate the effectiveness of the FPI layer through a simple experiment, where the layer is attached at the end of FlowNet [12]. The Flying Chairs dataset [12] was used with the original split which has 22,232 training and 640 test samples. In this case, the FPI layer plays the role of post-processing. We attached a very simple FPI layer consisting of conv/deconv layers and recorded the average end-point error (aEPE) per epoch, as shown in Figure 1. While the number of additional parameters was extremely small (less than 0.01%) and the computation time was nearly the same as for the original FlowNet, there was a noticeable improvement in performance with the FPI layer.

C. MULTI-LABEL CLASSIFICATION

The multi-label text classification dataset (Bibtex) is introduced in [16]. The goal of the task is to find the correlation between the data and the multi-label features. Both the data and the features are binary, with 1836 indicators and 159 labels, respectively. The number of positive indicators and labels differs for each data, and the number of true labels is unknown during the evaluation process. We used the same training and test split as in [16] and evaluated the F_1 scores. Here, we used two single FPI layer networks with FPI_GD and FPI_NN, respectively. We set g of the FPI_NN and f of the FPI_GD to similar structures that contain one or two fully-connected layers and activation functions. As mentioned, the detailed structures of the networks are described in the supplementary materials. Table 2 shows the F_1 scores ("GT" stands for "ground truth"). Here, DVN (adversarial) achieved the best performance, but it generated adversarial samples for data augmentation. Both FPI_GD layer and FPI_NN layer achieved better performance than DVN (GT) under the same conditions. Despite their simple structures, our algorithms had the best performance among those using only the training data, which confirms the effectiveness of the proposed method.

TABLE 2. F_1 score of multi-label text classification (higher is better). Our method shows the best performance among those using only the training data..

Method	F1 score
MLP [9]	38.9
Feedforward net [5]	39.6
SPEN [9]	42.2
ICNN [5]	41.5
DVN (GT) [8]	42.9
DVN (adversarial)[14]	**44.7**
FPI_GD layer (ours)	43.2
FPI_NN layer (ours)	43.4

V. CONCLUSION AND FUTURE WORK

This paper proposes a novel architecture using FPI as a layer of the neural network. The backward FPI layer is also proposed to backpropagate the FPI layer efficiently. We proved that both forward and backward FPI layers are guaranteed to converge under mild conditions. All the components were highly modularized so that we could efficiently apply the FPI layer to practical tasks by only changing the structure of g. Two representative cases of the FPI layer (FPI_NN and FPI_GD) were introduced. The experiments show that our method has advantages for several problems compared to the feedforward network. For problems such as denoising, the iterative structure of the FPI layer can be more helpful, while for other problems, it can be used to refine the performance of an existing method. Finally, we also demonstrate in the multi-label classification example that the FPI layer can achieve state-of-the-art performance with a very simple structure.

Since this research area has not been studied extensively, there is high potential for improvement. First, new structures of g and their applications are worth studying. As a simple and intuitive example, we can use another gradient-based algorithm such as Adam [18] instead of gradient descent in FPI_GD. Using multiple input sources can be an interesting direction for future research. We used a single input source x in this work but multiple input sources and alternate optimizations using multiple g may yield new effects. Another interesting direction for research is learning the initial value x_0, which was initialized to a zero matrix (or zero vector) in our study, based on z for improving the efficiency of the algorithms.

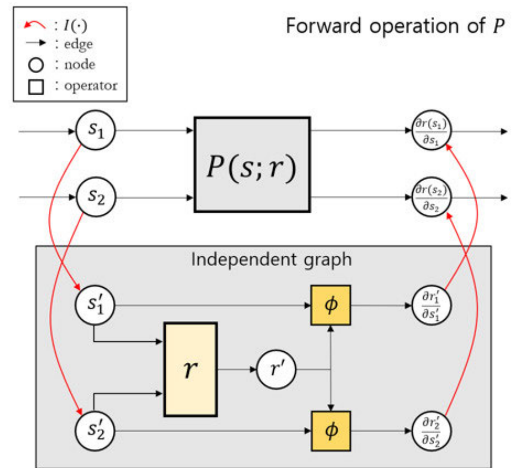

FIGURE 2. Forward operation of partial differentiation.

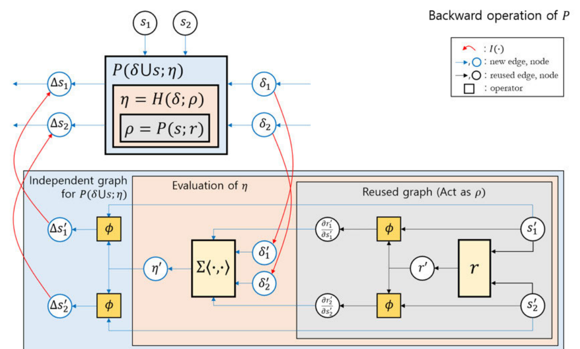

FIGURE 3. Backward operation of partial differentiation.

APPENDIX A
FIGURES OF PARTIAL DIFFERENTIATION AND THE BACKWARD FPI LAYER

The following figures (Figures 2 to 4) show the structures of the proposed partial differentiation operator and the backward FPI layer. Here, we can see that all the operations are highly modularized, which allow multiple differentiations in the usual autograd framework without any explicit Jacobian computations.

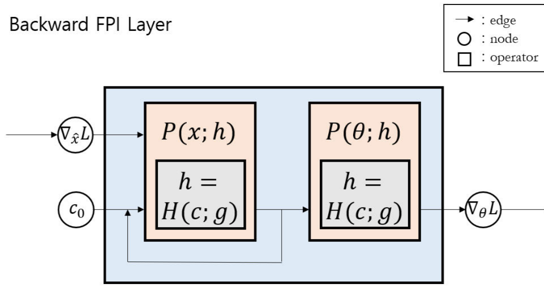

FIGURE 4. Backward FPI layer.

APPENDIX B
IMPLEMENTATION DETAILS

In all the experiments, we used the following criterion to determine the convergence of the FPI layer:

$$\beta = \frac{\|x_{n+1} - x_n\|^2}{\|x_n\|^2}$$

using the L_2-norm. When β fell below a certain threshold, x_{n+1} was considered to be converged, and we stopped the iteration. For all the experiments, we used two types of network modules for g of the FPI layer:

1) **The FPI_NN layer:** To see the full potential of the FPI layer, we tested an FPI layer with g as a general (small) neural network module. In this case, g can become an arbitrary function, and we can explore more diverse possibilities of the FPI layer. The only issue here is that the Lipschitz constant of g might not be bounded. Based on our empirical experience, using small initial weights for g was sufficient in our empirical experience.
2) **The FPI_GD layer:** Inspired by the EFNs [5], [9], [14], this layer performs a simple numerical optimization and yields the solution as the output of the layer. We define the energy function as a small neural network with a scalar output, and based on this energy function, g is defined to be a simple gradient descent step with a fixed step size. Unlike most existing EFNs, our version can perform backpropagation easily with the existing autograd functionalities.

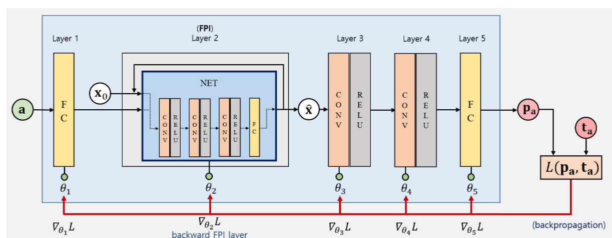

FIGURE 5. Example multi-layer network using an FPI layer (layer 2). Note that the shapes of input a and output $\hat{x}$ can be changed..

Figure 5 shows an example multi-layer network including an FPI_NN layer for a mini-batch size of 1. p_a is the prediction for the input data a, and t_a is the ground truth. As seen in the figure, the FPI layer is composed of a neural network module. When passing the FPI layer during the backpropagation process, the backward FPI layer is used to compute $\nabla_{\theta_2} L$.

For the experiments with image inputs, such as image denoising and optical flow, only the FPI_NN layer was tested. In order for the EFN of the FPI_GD layer to have a scalar output, we attached a mean-squared layer at the end of the network. For the multi-label classification, the performance was evaluated for both the FPI_NN and FPI_GD. Note that, for all the above layers, the fixed-point iteration variable x was concatenated with the layer input z and passed onto g (e.g., for vector inputs, $g(x, z; \theta) = g([x^T\ z^T]^T; \theta)$). Accordingly, the size of the input for g was greater than that of the output. x_0 was either a zero vector or a zero matrix in all the experiments.

All the training was performed using the Adam optimizer with a learning rate of 10^{-3}, and no weight decay was used. In the following experiments, we used the ReLU activation function most of the time. Although this does not exactly align with the assumptions in the paper, we used it nonetheless, based on common practices in deep learning and, confirmed that the FPI layer still performed well.

A. IMAGE DENOISING

The first 500 images of the Flying Chair dataset [12] were cropped to 180×180 around the center. Of these, the first 400 images were used to train the networks, and the latter 100 were used as test samples. All the images were converted to grayscale.

All the network modules consisted of two 2D convolution layers with 32 intermediate channels and a ReLU activation after the first convolution layer in the proposed method, and the baseline (non-FPI) feedforward network also shared the same structure. The channel sizes of the network's input and output were both one, since both input and output were grayscale images. All the models were trained for 20 epochs, and the final results were reported based on the best epoch for each method. Gradient clamping with a max-norm value of 0.1 was used to prevent the divergence of the FPI layers, and the convergence threshold of the FPI layer was set to 10^{-7}. The initial values of the network weights were set to ten times smaller than the default initialization of PyTorch [13]. Figure 6 shows the image denoising examples.

B. OPTICAL FLOW

We used the FlowNetS [12] structure for this experiment. As in [12], we tested the performance on the Flying Chair dataset with the same training and test split. The number of channels in FlowNetS starts at 6 and increases to 1024 using 10 conv layers, which are followed by several deconv layers. We attached an FPI layer at the end of FlowNetS, where g consists of one conv layer with four input and output channels and one deconv layer with four input and two output channels. The strides of the conv and deconv layers were set to two

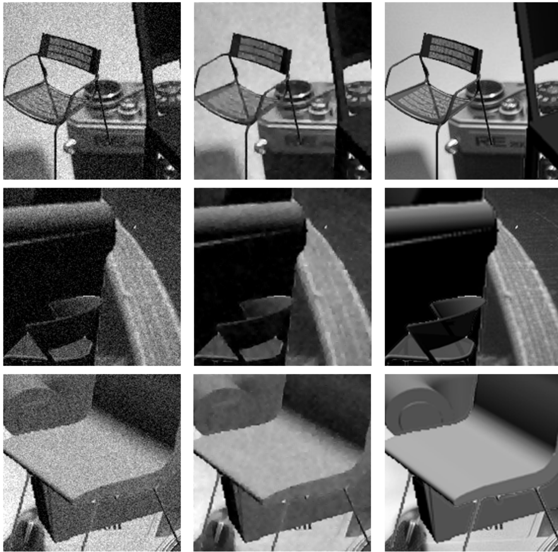

FIGURE 6. Image denoising examples. The left column shows the noisy input, and the right column is the ground truth. The denoised images are shown in the middle column.

for both downsampling and upsampling. The final output of the proposed model was the summation of the output of the FPI layer and that of the original FlowNetS. Note that both FlowNetS and the FPI layer were trained end-to-end in this experiment. The convergence threshold of the FPI layer was set to 10^{-13} and the initial values of the network weights were 100 times smaller than the default initialization of PyTorch. All other training settings were identical to those of the original FlowNet. Note that, for this experiment, the performance was worse than that of the original FlowNetS when a (non-FPI) feedforward module of the same structure was attached to the end of FlowNetS, and thus this result is omitted from the experimental results.

C. MULTI-LABEL CLASSIFICATION

For this experiment, all the training settings of the proposed methods were the same as in the other algorithms compared, except for the network structure. The network structure for FPI_NN was composed of two fully-connected (FC) layers with 512 hidden nodes, a ReLU activation after the first FC layer, and an additional sigmoid layer after the second FC layer to normalize the output between zero and one. In this case, the convergence threshold was 10^{-8}. For FPI_GD, the energy function had only one FC layer with ReLU activation and a mean-squared term to obtain a scalar output. Here, the number of hidden nodes was also 512, and an additional sigmoid layer was added after the FPI_GD layer. We set the step size to 1.0 for the gradient descent in FPI_GD and used a different convergence criterion as follows:

$$\beta' = \|x_{n+1} - x_n\|^2$$

where the convergence threshold was 10^{-4}. The input and output sizes of both networks were 1836 and 159, respectively, which is equal to the numbers of indicators and labels.

APPENDIX C
ADDITIONAL EXPERIMENTS: A CONSTRAINED PROBLEM

Here, we show the feasibility of our algorithm by solving a constrained optimization problem ($\min_x \|x - a\|^2$) with a box constraint ($-\mathbf{1} \leq x \leq \mathbf{1}$). The goal of the problem is to learn the functional relation based on training samples (a, t), where t is the ground-truth solution. We used single FPI layer networks for this problem.

Performance was evaluated for the FPI_NN network, the FPI_GD network, and a non-FPI network that has the same structure as g of the FPI_NN. The structures of g of FPI_NN and the energy function of FPI_GD were both linear-ReLU-linear. The dimensions of a, x and the number of hidden nodes were 10, 10, and 32, respectively. We randomly generated a using zero-mean Gaussian distributions. 10,000 training samples were generated with $\sigma = 2$, and 1,000 test samples with $\sigma = 1$. The convergence threshold was set to 10^{-6} for both FPI_NN and FPI_GD layers and the step size of the gradient descent in the FPI_GD was fixed to 0.01. All the models were trained for 40 epochs with a batch size of 100, and the training and test losses (MSE) were reported.

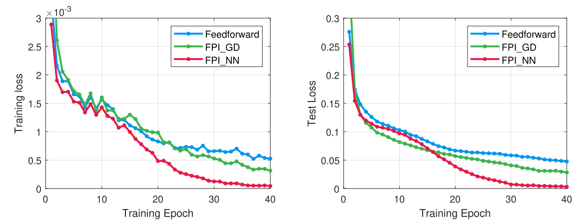

FIGURE 7. Training and test losses per epoch in the constrained problem. The FPI_GD and FPI_NN networks performed better than the feedforward network with the same number of parameters..

Figure 7 shows the training and test losses per epoch. Here, we can see that the FPI_NN outperformed the other networks in both training and test losses.

APPENDIX D
ADDITIONAL LEMMAS FOR THE CONVERGENCE OF BACKWARD FPI LAYER

Here, we use an arbitrary norm metric for all the vector and matrix norms. The following lemma holds for vectors x and b, matrix A, and scalar $k < 1$:

Lemma 1: If the matrix norm $\|A\| < 1$, then the linear transformation by weight A, i.e., $f(x) = Ax + b$, is a contraction mapping.

Proof:

$$\frac{\|f(x_1) - f(x_2)\|}{\|x_1 - x_2\|} = \frac{\|Ax_1 - Ax_2\|}{\|x_1 - x_2\|} = \frac{\|A(x_1 - x_2)\|}{\|x_1 - x_2\|} \leq \frac{\|A\| \cdot \|x_1 - x_2\|}{\|x_1 - x_2\|} = \|A\| \leq k < 1$$

By the definition, f is a contraction mapping. □

In section 3.4, $\|J_g(\hat{x})\| \leq k < 1$, which means that the linear transformation with weight matrix $J_g(\hat{x})$ is a contraction

mapping by *Lemma 1*. Therefore, (10) of the backward FPI in the main paper is a contraction mapping.

ACKNOWLEDGMENT

(Younghan Jeon and Minsik Lee are co-first authors.)

REFERENCES

[1] A. Agrawal, B. Amos, S. Barratt, S. Boyd, D. Diamond, and S. Kolter, J. Zico, "Differentiable convex optimization layers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 9558–9570.

[2] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi, "Differentiating through a cone program," 2019, *arXiv:1904.09043*. [Online]. Available: http://arxiv.org/abs/1904.09043

[3] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," in *Proc. Int. Conf. Neural Netw.*, 1987, pp. 609–618.

[4] B. Amos, S. Kolter, and J. Zico, "Optnet: Differentiable optimization as a layer in neural networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 136–145.

[5] B. Amos, L. Xu, and J. Kolter, "Input convex neural networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 146–155.

[6] Bai, Shaojie, Kolter, J Zico, and Koltun, Vladlen, "Deep equilibrium models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 688–699.

[7] S. Banach, "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales," *Fundam. Math.*, vol. 3, pp. 133–181, Dec. 1922.

[8] Beardsell, Philippe and Hsu, Chih-Chao. (2020). *Structured Prediction with Deep Value Networks*. [Online]. Available: https://github.com/philqc/deep-value-networks-pytorch

[9] Belanger, David and McCallum, Andrew, "Structured prediction energy networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 983–992.

[10] Burden, Richard and Faires, *Numerical Analysis*. Boston, MA, USA: Cengage Learning, 2004.

[11] Djolonga, Josip and Krause, Andreas, "Differentiable learning of submodular models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1013–1023.

[12] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazárbaá, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "FlowNet: Learning optical flow with convolutional networks," 2015, *arXiv:1504.06852*. [Online]. Available: http://arxiv.org/abs/1504.06852

[13] Glorot, Xavier and Bengio, Yoshua, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[14] M. Gygli, M. Norouzi, and A. Angelova, "Deep value networks learn to evaluate and iteratively refine structured outputs," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 1341–1351.

[15] Jiang, Borui, Luo, Ruixuan, Mao, Jiayuan, Xiao, Tete, and Jiang, Yuning, "Acquisition of localization confidence for accurate object detection," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 784–799.

[16] Katakis, Ioannis, Tsoumakas, Grigorios, and Vlahavas, Ioannis, "Multilabel text classification for automated tag suggestion," in *Proc. ECML/PKDD*, vol. 18, 2008, p. 5.

[17] M. A. Khamsi and W. A. Kirk, *An Introduction to Metric Spaces Fixed Point Theory*, vol. 53. Hoboken, NJ, USA: Wiley, 2011.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[19] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, "A tutorial on energy-based learning," *Predicting Struct. Data*, vol. 1, p. 10, Aug. 2006.

[20] R. Liao, Y. Xiong, E. Fetaya, L. Zhang, L. Yoon, X. Pitkow, R. Urtasun, and R. Zemel, "Reviving and improving recurrent back-propagation," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3082–3091.

[21] Y. Peng, B. Deng, J. Zhang, F. Geng, W. Qin, and L. Liu, "Anderson acceleration for geometry optimization and physics simulation," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–14, 2018.

[22] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Phys. Rev. Lett.*, vol. 59, no. 19, p. 2229, 1987.

[23] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proc. 21st Int. Conf. Mach. Learn. (ICML)*, 2004, p. 104.

[24] P.-W. Wang, P. L. Donti, B. Wilder, and Z. Kolter, "SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver," 2019, *arXiv:1905.12149*. [Online]. Available: http://arxiv.org/abs/1905.12149

YOUNGHAN JEON (Member, IEEE) received the bachelor's degree in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2014. He is currently pursuing the Ph.D. degree in electrical and computer engineering. His current research interests include deep learning architecture, optimization, and recurrent networks.

MINSIK LEE (Member, IEEE) received the B.S. and Ph.D. degrees from the School of Electrical Engineering and Computer Science, Seoul National University, South Korea, in 2006 and 2012, respectively. From 2012 to 2013, he was a Postdoctoral Researcher with the School of Electrical Engineering and Computer Science. In 2014, he joined Seoul National University as a BK21 Assistant Professor. He is currently an Associate Professor with Hanyang University, Ansan, South Korea. His research interests include shape and motion analysis, deformable models, computer vision, deep learning, pattern recognition, and their applications.

JIN YOUNG CHOI (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in control and instrumentation engineering from Seoul National University, Seoul, South Korea, in 1982, 1984, and 1993, respectively. From 1984 to 1989, he joined the Project of TDX Switching System, Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. From 1992 to 1994, he was with the Basic Research Department, ETRI. Since 1994, he has been with Seoul National University. From 1998 to 1999, he was a Visiting Professor with the University of California at Riverside, Riverside, CA, USA. He is currently a Professor with the School of Electrical Engineering, Seoul National University, where he is also with the Engineering Research Center for Advanced Control and Instrumentation, the Automatic Control Research Center, and the Automation and Systems Research Institute. His current research interests include adaptive and learning systems, visual surveillance, motion pattern analysis, object detection and tracking, and pattern learning and recognition. He was a Senior Member of Technical Staff involved in the neural information processing system with ETRI.