
Differentiable Fixed-Point Iteration Layer

Younghan Jeon^{* 1} Minsik Lee^{* 2} Jin Young Choi¹

Abstract

Recently, several studies proposed methods to utilize some restricted classes of optimization problems as layers of deep neural networks. However, these methods are still in their infancy and require special treatments, i.e., analyzing the KKT condition, etc., for deriving the backpropagation formula. Instead, in this paper, we propose a method to utilize fixed-point iteration (FPI), a generalization of many types of numerical algorithms, as a network layer. We show that the derivative of an FPI layer depends only on the fixed point, and then we present a method to calculate it efficiently using another FPI which we call the backward FPI. The proposed method can be easily implemented based on the autograd functionalities in existing deep learning tools. Since FPI covers vast different types of numerical algorithms in machine learning and other fields, it has a lot of potential applications. In the experiments, the differentiable FPI layer is applied to two scenarios, i.e., gradient descent iterations for differentiable optimization problems and FPI with arbitrary neural network modules, of which the results demonstrate the simplicity and the effectiveness.

1. Introduction

Fixed-point iteration (FPI) has been one of the most important building blocks in many areas of machine learning for a long time. It usually appears in the form of numerical optimization, etc., to find desired solutions for given problems. There are tons of examples in the literature; support vector machines, expectation-maximization, compressed sensing, value/policy iterations in reinforcement learning, and the

list goes on. All of these are popular examples that shaped the field for the past few decades.

However, such existing techniques are becoming less used in the current deep learning era, due to the superior performance of deep neural networks. Even if there already exists a more traditional algorithm that is well-defined, using it directly as a part of a new machine learning algorithm with a deep network can be tricky. Unless the algorithm gives a closed-form solution or we are to use the algorithm separately as preprocessing or post-processing, incorporating it in a deep network is not always straightforward because the usual iterative process makes backpropagation difficult. A naïve way to resolve this issue is to regard each iteration as a node in the computational graph, but this requires a lot of computational resources. As a result, it is not an exaggeration that many of the current day works are rewriting a large part of the existing methodologies with deep networks.

Recently, several papers proposed using certain types of optimization problems as layers in deep networks (Belanger & McCallum, 2016; Amos & Kolter, 2017; Amos et al., 2017). In these approaches, the input or the weights of the layers are used to define the cost functions of the optimization problems, and the solutions of the problems become the layers' output. For certain classes of optimization problems, these layers are differentiable. These methods can be used to introduce a prior in a deep network and provide a possibility of bridging the gap between deep learning and some of the above traditional methods. However, they are still premature and require non-trivial efforts to implement in actual applications. Especially, the backpropagation formula has to be derived explicitly for each different formulation based on some criteria like the KKT conditions, etc. This limits the practicality of the approaches since there can be numerous different optimization formulations depending on the actual problems.

In this paper, we instead focus on FPI which is the basis of many numerical algorithms including most gradient-based optimizations. In the proposed FPI layer, the layer's input or its weights are used for defining 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

^{*}Equal contribution ¹Department of Electrical and Computer Engineering, ASRI, Seoul National University, South Korea ²Division of Electrical Engineering, Hanyang University, South Korea. Correspondence to: Jin Young Choi <jychoi@snu.ac.kr>.

much more efficient than adding all the individual iterations to the computational graph. However, an even more important advantage of the proposed FPI layer is that the derivative can be easily calculated based on another independent computational graph that describes a single iteration of the update equation. In other words, we do not need a separate derivation for the backpropagation formula and can utilize the autograd functionalities in existing deep learning tools. This makes the proposed method very simple and practical to use in various applications.

Specifically, the derivative of the FPI layer in its original form requires to calculate the Jacobian of the update equation. Since this can be a computational burden, we show that this computation can be transformed to another equivalent FPI, which is called the *backward FPI* in this paper. We also show that if the aforementioned conditions for the FPI layer hold, then the backward FPI also converges. In summary, both the forward and backward processes are composed of FPIs in the proposed method.

Since FPI covers many other types of numerical algorithms as well as optimization problems, there are a lot of potential applications for the proposed method. Contributions of the paper are summarized as follows.

- We propose a novel layer based on FPI. The FPI layer can perform similar functionalities as existing layers based on differentiable optimization problems, but the implementation is much simpler and the backpropagation formula can be universally derived.
- We show that under mild conditions, the FPI layer is differentiable and the derivative depends only on the fixed point, which eliminates the necessity of constructing a large computational graph.
- To reduce the memory consumption in backpropagation, we derive the backward FPI that can calculate the derivative with a reduced memory requirement. Under the same conditions mentioned in the previous item, it is guaranteed that the backward FPI converges.

The remainder of this paper is organized as follows: We first introduce the related work in Section 2. The proposed FPI layer is explained in Section 3, the experimental results follow in Section 4. Finally, we conclude the paper in Section 5.

2. Background and Related Work

Fixed-point iteration: For a given function g , the fixed-point iteration is defined based on the following update equation:

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

where $\{x_n \in \mathbb{R}^d\}$ is a sequence of vectors. If x_n converges to some \hat{x} , then it is called a fixed point of g . The gradient descent method ($x_{n+1} = x_n - \gamma \nabla f(x_n)$) and Newton’s method ($x_{n+1} = x_n - f(x_n)/f'(x_n)$ for a scalar sequence) are popular examples of fixed-point iteration. Many numerical algorithms are based on fixed-point iteration, and there are also many popular examples in machine learning.

Here are some important concepts about fixed-point iteration.

Definition 1 (Contraction mapping). (Khamsi & Kirk, 2011) On a metric space (X, d) , function $f : X \rightarrow X$ 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 .

$$d(f(x_1), f(x_2)) \leq k \cdot d(x_1, x_2). \quad (2)$$

The smallest k that satisfies the above condition is called the Lipschitz constant of f . We use the most common L_2 distance metric in this paper.

Based on the above definition, the Banach fixed-point theorem (Banach, 1922) states the following.

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 to estimate the energy (or error) functions have generated considerable recent research interest. These energy networks have a different structure from the general feed-forward neural networks, and the concept was first proposed in LeCun et al. 2006. They predict the answer by the input which minimizes the network’s output.

The structured prediction energy networks (SPENs) (Belanger & McCallum, 2016) perform gradient descent on an energy function network to find the solution, and an SSVM (Tsochantaridis et al., 2004) loss is defined based on the solution to train the network. The input convex neural networks (ICNNs) (Amos et al., 2017) are defined in a specific way so that the networks have convex structures with respect to (w.r.t.) the input, and its learning and inference are performed by the bundle entropy method which is derived based on the KKT optimality conditions. The deep value networks (Gygli et al., 2017) and the IoU-Net (Jiang et al., 2018) directly learns the loss metric such as the intersection over union (IoU) of bounding boxes then perform inference by gradient based optimization methods. However, these methods generally require complex learn-

ing processes and each method is specialized to a limited range of applications. Also, they require various approximations/relaxations or constraints such as bounded condition.

On the other hand, the end-to-end SPENs (Belanger et al., 2017) directly backpropagates the whole gradient-based inference process that has a fixed number of gradient steps. Although this alleviates the need of a complicated derivation for backpropagation, the memory requirement increases as the number of steps increases since all the steps are constructed in a computational graph and the gradient of each step must be calculated. If the number of steps is small, there is a high possibility of not converging to the optimal solution, which is not ideal in the perspective of energy-based learning.

Although the above approaches provide novel ways of utilizing neural networks in optimization frameworks, there is no apparent way to combine them within other existing deep networks. Moreover, they are mostly limited to a certain type of problems and require complicated learning processes.

On the other hand, the proposed method can be applied in broader situations than these approaches, and these approaches can be equivalently implemented with the proposed method once the update equation for the optimization problem is derived.

Differentiable optimization layers:

Recently, a few papers using optimization problems as a layer of deep learning architecture have been proposed. Such a structure can contain a more complicated behavior in one layer than a feed-forward net, and can potentially reduce the depth of the network.

OptNet (Amos & Kolter, 2017) presented how to use the quadratic program (QP) as a layer of neural networks. They also uses the KKT conditions to compute the derivative of the solution of QP. Agrawal et al. 2019a proposed an approach to differentiate disciplined convex programs which is a subclass of convex optimization problems. There are a few other researches trying to differentiate optimization problems such as submodular models (Djoulonga & Krause, 2017), cone program (Agrawal et al., 2019b), semidefinite program (Wang et al., 2019), and so on. However, most of them has limited applications and users need to adapt their problems to the rigid problem settings. On the other hand, our method makes it easy to use a large class of iterative algorithms as a network layer, which also includes the differentiable optimization problems.

3. Proposed Method

The fixed-point iteration formula contains a wide variety of forms and can be applied to most iterative algorithms. Section 3.1 describes the basic structure and principles of the FPI layer. Section 3.2 and 3.3 explains the differentiation of the layer for backpropagation. Section 3.5 presents some example applications. Section 3.4 describes the convergence of the backward FPI.

3.1. Structure of the fixed-point iteration layer

Here we describes 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 also has a Lipschitz constant less than one for x , and the following fixed point iteration converges to a unique point according to the Banach fixed-point iteration theorem:

$$x_{n+1} = g(x_n, z; \theta), \quad (3)$$

$$\hat{x} = \lim_{n \rightarrow \infty} x_n. \quad (4)$$

In practice, g could be a neural network or an algorithm like gradient descent. The FPI layer can be defined based on the above relations. The FPI layer \mathbf{F} takes data or output of the previous layer as input z , and yields the fixed point of g as the layer's output:

$$\hat{x} = \mathbf{F}(x_0, z; \theta). \quad (5)$$

Here, θ acts as the parameter of the layer. Here, we can notice that 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. This will also be confirmed later in the derivation of backpropagation. Hence, x_0 can be predetermined to any value. Accordingly, we will often express \hat{x} as a function of z and θ , i.e., $\hat{x}(z; \theta)$.

When using an FPI layer, (3) is actually repeated until convergence to find the output \hat{x} . We may use some acceleration techniques such as the Anderson acceleration (Peng et al., 2018) if it takes too long to converge.

In multi-layer networks, z from the previous layer is passed onto the FPI layer, and its output can be passed to another layer to continue the feed-forward process. 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 can be different.

3.2. Differentiation of the FPI layer

Same as the other network layers, learning of \mathbf{F} is performed by updating θ based on backpropagation. For this, the derivatives of the FPI layer has to be calculated. One

simple way to compute the gradients is to construct a computational graph for all the iterations up to the fixed point \hat{x} . For example, if it converges in N iterations ($x_N = \hat{x}$), all the derivatives from x_0 to x_N can be calculated by the chain rule. However, this method is not only time consuming but also requires a lot 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 convergence are actually not needed in the computation of the derivatives. Hence, we can only retain the value of \hat{x} to perform backpropagation, and consider the entire \mathbf{F} as a single node in the computational graph. Here, we provide the derivation of $\partial\hat{x}/\partial\theta$, and that of $\partial\hat{x}/\partial z$ (which is needed for the backpropagation of layers before \mathbf{F}) are mostly similar.

Note that the following equation is satisfied at the fixed point \hat{x} :

$$\hat{x} = g(\hat{x}, z; \theta). \quad (6)$$

If we differentiate both sides of the above equation w.r.t. θ , we have

$$\frac{\partial\hat{x}}{\partial\theta} = \frac{\partial g}{\partial\theta}(\hat{x}, z; \theta) + \frac{\partial g}{\partial x}(\hat{x}, z; \theta) \frac{\partial\hat{x}}{\partial\theta}. \quad (7)$$

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

$$\frac{\partial\hat{x}}{\partial\theta} = \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{-1} \frac{\partial g}{\partial\theta}(\hat{x}, z; \theta), \quad (8)$$

which confirms the fact that the derivative of the output of $\mathbf{F}(x_0, z; \theta) = \hat{x}$ depends only on the value of \hat{x} .

A nice thing about the above derivation is that it only requires the derivatives of a single iteration g . This can be easily calculated based on the autograd functionalities of existing deep learning tools, and there is no need for a separate derivation. However, care should be taken about the fact that the differentiations w.r.t. x and θ are *partial* differentiations. x and θ might have some dependency with each other in a usual autograd framework, hence we have to create an independent computational graph that having x and θ as leaf variables cloned from the nodes in the original computational graph. In this way, we can perform the partial differentiation accurately.

One downside of the above derivation is that it requires to calculate the Jacobians of g , which may need a lot of memory space. In the next section, we will provide an efficient way to resolve this issue with another FPI.

3.3. Backward fixed-point iteration

Here, we assume that the output of \mathbf{F} is passed onto some other layers for more processing, and then there is a loss

function at the end of the network. If we summarize all the layers and the loss function after the FPI layer into a single function L , then what we need for backpropagation are $\nabla_{\theta}L(\hat{x})$ and $\nabla_zL(\hat{x})$. Similar to the previous section, we will only provide the derivation of $\nabla_{\theta}L(\hat{x})$.

According to (8), we have

$$\begin{aligned} \nabla_{\theta}L &= \left(\frac{\partial\hat{x}}{\partial\theta} \right)^{\top} \nabla_{\hat{x}}L \\ &= \left(\frac{\partial g}{\partial\theta}(\hat{x}, z; \theta) \right)^{\top} \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{-\top} \nabla_{\hat{x}}L \end{aligned} \quad (9)$$

This section describes how to calculate the above equation efficiently. (9) can be divided into two steps as follows:

$$c = \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{-\top} \nabla_{\hat{x}}L, \quad (10)$$

$$\nabla_{\theta}L = \left(\frac{\partial g}{\partial\theta}(\hat{x}, z; \theta) \right)^{\top} c. \quad (11)$$

Rearranging (10) yields the following results.

$$c = \left(\frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{\top} c + \nabla_{\hat{x}}L. \quad (12)$$

Note that the above equation has the form of FPI, hence we perform the following FPI which is called the backward FPI:

$$c_{n+1} = \left(\frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{\top} c_n + \nabla_{\hat{x}}L. \quad (13)$$

c can be obtained by initializing c to some arbitrary value and repeating the above update until convergence. If the forward iteration g is a contraction mapping, we can prove that the backward FPI is also a contraction mapping, which is guaranteed to converge to a unique point. The proof of convergence is discussed in detail in the next section.

In the above update equation, $\left(\frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{\top} c_n$ can be calculated with explicitly calculating the Jacobian. If we define a new function h as

$$h(x, z, c; \theta) = c^{\top} g(x, z; \theta), \quad (14)$$

then (13) becomes

$$c = \frac{\partial h}{\partial x}(\hat{x}, z, c; \theta) + \nabla_{\hat{x}}L. \quad (15)$$

Note that the output of h is scalar. Here, we can consider h as another small network containing only one step of

FPI(g). We can also easily compute the gradient of h based on the autograd functionalities.

Similarly, the second step (11) is also expressed using h :

$$\nabla_{\theta} L = \frac{\partial h}{\partial \theta}(\hat{x}, z, c; \theta). \quad (16)$$

In this way, we can compute $\nabla_{\theta} L$ without any memory-intensive operations.

3.4. Convergence of the fixed-point iteration 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. In practice, we empirically found out that setting the weights (θ) to small values is enough for making g a contraction mapping throughout the training procedure.

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

Proposition 1. *If g is a contraction mapping, the backward FPI (Eq. (13)) converges to a unique point.*

Proof. For simplicity, we omit z and θ from g . By the definition of the contraction mapping and since we assume the metric is the L2 distance,

$$\frac{\|g(x_2) - g(x_1)\|}{\|x_2 - x_1\|} \leq k \quad (17)$$

for all x_1 and x_2 ($0 \leq k < 1$). For a unit vector v and a scalar t , let $x_2 = x_1 + tv$. Then, Eq. (17) is,

$$\frac{\|g(x_1 + tv) - g(x_1)\|}{|t|} \leq k. \quad (18)$$

For another unit vector u ,

$$\frac{u^{\top}(g(x_1 + tv) - g(x_1))}{|t|} \leq \frac{\|g(x_1 + tv) - g(x_1)\|}{|t|} \leq k \quad (19)$$

which indicates that

$$\lim_{t \rightarrow 0^+} \frac{u^{\top}(g(x_1 + tv) - g(x_1))}{|t|} = \nabla_v(u^{\top}g)(x_1) \leq k \quad (20)$$

According to the chain rule, $\nabla(u^{\top}g) = (u^{\top}J_g)^{\top}$ where J_g is the Jacobian of g .

$$\begin{aligned} \nabla_v(u^{\top}g)(x_1) &= (\nabla(u^{\top}g)(x_1))^{\top} \cdot v \\ &= u^{\top}J_g(x_1)v \leq k \end{aligned} \quad (21)$$

Let $x_1 = \hat{x}$ then $u^{\top}J_g(\hat{x})v \leq k$ for all u, v that satisfy $\|u\| = \|v\| = 1$. Therefore, $\|J_g(\hat{x})\| \leq k < 1$ which means the linear mapping by weight $J_g(\hat{x})$ is a contraction mapping. By the Banach fixed-point theorem, backward FPI converges to the unique fixed-point. \square

3.5. Examples

3.5.1. GRADIENT DESCENT FPI LAYER

A perfect example for using the FPI layer is the gradient descent method. Energy function networks are scalar-valued networks which estimate the energy (or error) functions. Unlike a typical network which obtains the prediction directly from the output of the network ($pred_a = f(a; \theta)$), the energy network gets the answer by optimizing the input variables for the network ($pred_a = \operatorname{argmin}_x f(x, a; \theta)$). We can optimize the network f by gradient descent.

$$x_{n+1} = x_n - \gamma \nabla f(x_n, a; \theta) \quad (22)$$

This is a form of FPI and the fixed point \hat{x} is the optimal point of f .

$$\hat{x} = \operatorname{argmin}_x f(x, a; \theta) \quad (23)$$

In the case of a single FPI layer network, the loss function is expressed as follows.

$$\min_{\theta} L(x^*, \hat{x}) \quad (24)$$

This exactly matches the objectives of energy function networks like ICNN. Therefore, the energy function networks can be effectively learned based on the proposed FPI layer. Moreover, we can build large networks by attaching different layers before and after the FPI layer, which becomes a large network containing an energy network as a component. Of course, using multiple FPI layers in a single network is also possible.

3.5.2. NEURAL NET FPI LAYER

It is also possible that g itself is a neural network module. The input variable recursively enters the same network and this is repeated until convergence. It can be much faster than the gradient descent FPI layer since it computes the output directly. This allows the FPI layer perform more complicated behaviors than using g directly without any FPI.

Figure 1 shows an example multi-layer network for mini-batch size 1. p_a is the prediction for the input data a and t_a is the ground truth. As can be seen in the figure, the FPI layer composed of a neural network module.

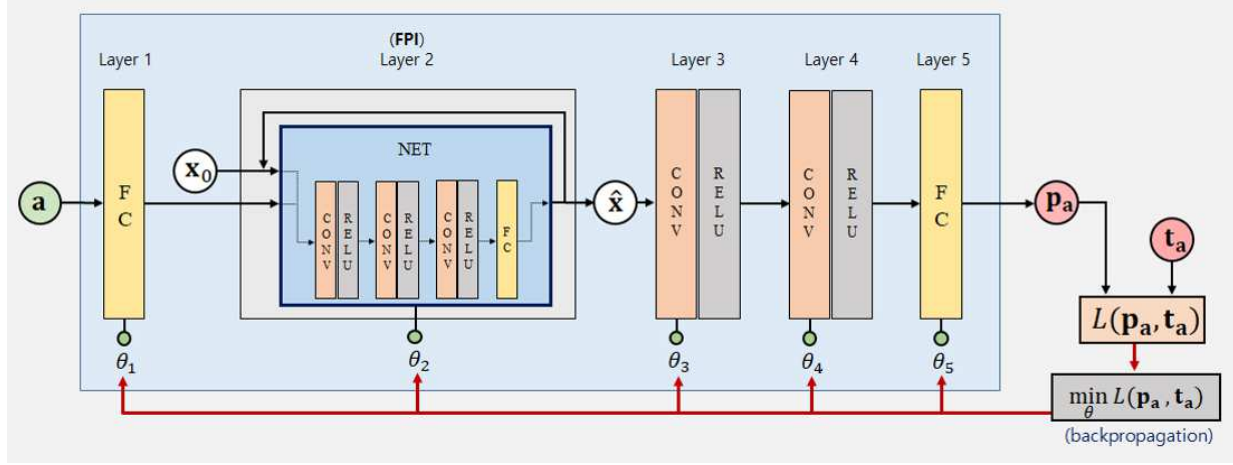


Figure 1. Example multi-layer network using neural net FPI layer.

4. Experiments

We show four experiments showing that the FPI layer can be used in various settings. In the toy example and image denoising problems, we compare the performance of the FPI layer to a non-FPI network that has the same structure with g . In the optical flow problem, a relatively very small FPI layer is attached at the end of FlowNet (Fischer et al., 2015) to show its effectiveness. Results on the multi-label classification problem shows that the FPI layer is superior in performance compared to the existing state-of-the-art algorithms.

4.1. Toy example: a constrained problem

We show the feasibility of our algorithm by learning a constrained optimization problem ($\min_x \|x - a\|^2$) with a box constraint ($-1 \leq x \leq 1$). Here, the inequalities are element-wise. The goal of the problem is to learn the functional relation based on training samples (a, t) , where t is the ground truth solution of the problem.

Figure 2 and 3 shows the train and test loss per epoch of FPI_GD layer, FPI_NN layer and the feedforward net which has the same structure as the FPI layers.

4.2. Image denoising

Here we compare the image denoising performance for gray images perturbed by Gaussian noise with variance σ^2 . To generate the image, we crop the flying chair dataset (Fischer et al., 2015) and convert it to gray scale (400 images for training and 100 images for testing). We used the neural network FPI layer and the feedforward network which has the same structure consists of convolution and ReLU layers. Peak signal-to-noise ratio (PSNR) is reported in Table 1 which is the most widely used measure of image

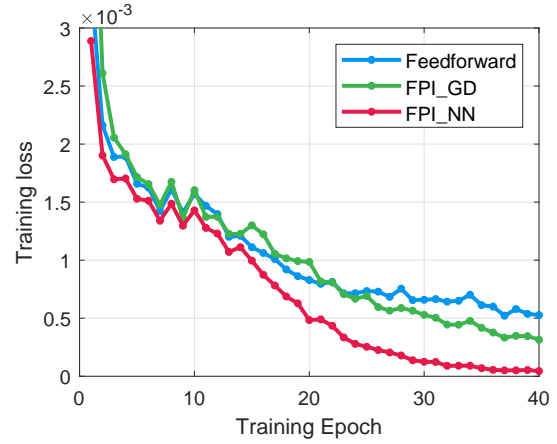


Figure 2. Train loss per epoch in constrained problem.

denoising.

Method	$\sigma = 15$	$\sigma = 20$	$\sigma = 25$
Feedforward net	32.18	30.44	29.09
FPI layer	32.43	31.00	29.74

Table 1. Denoising results (PSNR, higher is better)

Table 1 shows the FPI layer outperforms the feedforward net in all experiments. Note that the performance gap between the two algorithms is larger for more difficult tasks. This shows that the FPI layer is suitable for solving complex problems.

4.3. Optical flow

Optical flow is one of the major research areas of computer vision which aims to acquire motions by matching pixels in two image. We show that the FPI layer has a good effect by

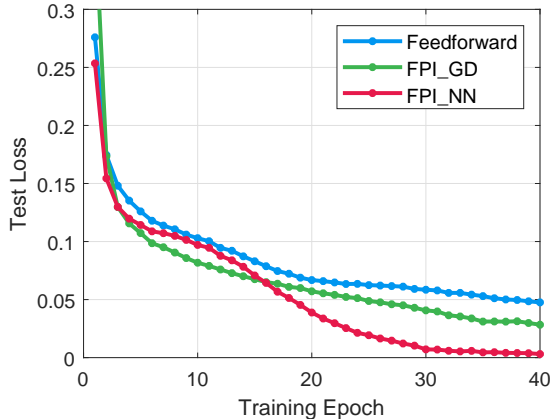


Figure 3. Test loss per epoch in constrained problem.

a simple experiment of attaching it at the end of the flownet (Fischer et al., 2015). In this case, the FPI layer plays a role as post processing. We attach a very simple FPI layer consists of one conv/deconv layer and record the average end point error (*aEPE*) per epoch in Figure 5.

Although the computation time is nearly same and the addition of the number of variables is extremely small, it shows a noticeable performance difference.

4.4. Multi label classification

Multi-label text classification dataset (Bibtex) was introduced by Katakis et al. 2008. Goal of the task is to find the correlation between the data and multi-labeled features. Both data and features are binary with 1836 indicators and 159 labels, respectively. The number of indicators and labels differs for each data, and the number of labels is not known during the evaluation process. We follow the same train an test split from Katakis et al. 2008 and report the F_1 scores.

Here we use the single FPI layer network for both gradient descent FPI (FPI_GD) and neural network FPI (FPI_NN) which has only one fully-connected hidden layer and ReLU activation. To format the output, FPI_NN has a 159 fully-connected layer equal to the number of labels, and FPI_GD has a mean-square term to make the output scalar. For experiments, we only need to specify the threshold of the convergence condition. There are no data-specific hyperparameters.

Table 2 shows the F_1 score on Bibtex multi-label classification dataset (higher is better). DVN(Adversarial) uses data augmentation to generate adversarial samples. Despite its simple structure, our algorithm performs the best among algorithms using only training data and their ground truth (GT).

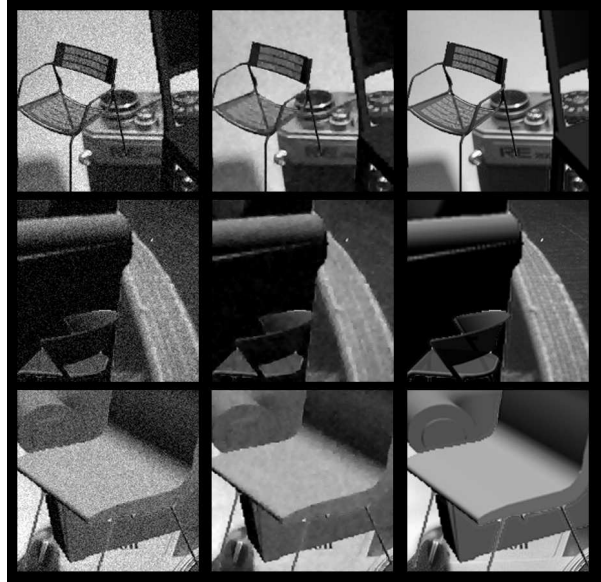


Figure 4. Example image denoising results. Left column shows the noisy input and right column is the ground truth. Denoised images are shown in the middle.

Method	F1 score
MLP (Belanger & McCallum, 2016)	38.9
Feedforward net (Amos et al., 2017)	39.6
SPEN (Belanger & McCallum, 2016)	42.2
ICNN (Amos et al., 2017)	41.5
DVN(GT)	42.9
DVN(Adversarial)(Gygli et al., 2017)	44.7
FPI_GD layer (Ours)	43.2
FPI_NN layer (Ours)	43.4

Table 2. F_1 score on multi-label text classification.

5. Conclusion

This paper proposed a novel architecture that uses the fixed-point iteration as a layer of the neural network. We derived the differentiation of the FPI layer using derivatives of the convergence (fixed) point. We also presented a method to compute the differentiation efficiently by another fixed-point iteration called backward FPI. Applications for two representative cases of FPI (gradient descent and neural network) are introduced, and there are various applications according to the form of FPI layer. Experiments show the potential power of the FPI layer compared to the feedforward network which has the same structure. It also shows that the FPI layer can be easily applied to various fields, and the performance is comparable with the state-of-the-art algorithms without the need for data specific hyperparameters.

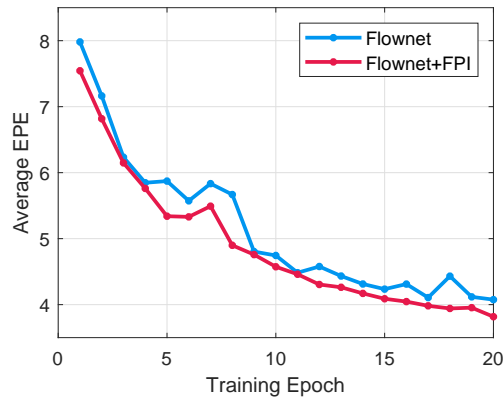


Figure 5. Average EPE per epoch (lower is better).

References

- Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, pp. 9558–9570, 2019a.
- Agrawal, A., Barratt, S., Boyd, S., Busseti, E., and Moursi, W. M. Differentiating through a conic program. *arXiv preprint arXiv:1904.09043*, 2019b.
- Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 136–145. JMLR. org, 2017.
- Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 146–155. JMLR. org, 2017.
- Banach, S. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math*, 3(1):133–181, 1922.
- Belanger, D. and McCallum, A. Structured prediction energy networks. In *International Conference on Machine Learning*, pp. 983–992, 2016.
- Belanger, D., Yang, B., and McCallum, A. End-to-end learning for structured prediction energy networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 429–439. JMLR. org, 2017.
- Djolonga, J. and Krause, A. Differentiable learning of sub-modular models. In *Advances in Neural Information Processing Systems*, pp. 1013–1023, 2017.
- Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., Van der Smagt, P., Cremers, D., and Brox, T. FlowNet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.
- Gygli, M., Norouzi, M., and Angelova, A. Deep value networks learn to evaluate and iteratively refine structured outputs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1341–1351. JMLR. org, 2017.
- Jiang, B., Luo, R., Mao, J., Xiao, T., and Jiang, Y. Acquisition of localization confidence for accurate object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–799, 2018.
- Katakis, I., Tsoumakas, G., and Vlahavas, I. Multilabel text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD*, volume 18, pp. 5, 2008.
- Khamsi, M. A. and Kirk, W. A. *An introduction to metric spaces and fixed point theory*, volume 53. John Wiley & Sons, 2011.
- LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.
- Peng, Y., Deng, B., Zhang, J., Geng, F., Qin, W., and Liu, L. Anderson acceleration for geometry optimization and physics simulation. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.
- Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 104, 2004.
- Wang, P.-W., Donti, P. L., Wilder, B., and Kolter, Z. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv preprint arXiv:1905.12149*, 2019.