Ph.D. DISSERTATION

Differentiable Fixed-Point Iteration Layer and Its Applications to Machine Learning Tasks

미분 가능한 고정점 반복 레이어 및 기계학습 문제에 대한 적용

BY

Younghan Jeon

AUGUST 2021

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Differentiable Fixed-Point Iteration Layer and Its Applications to Machine Learning Tasks

미분 가능한 고정점 반복 레이어 및 기계학습 문제에 대한 적용

지도교수 최 진 영

이 논문을 공학박사 학위논문으로 제출함

2021 년 8 월

서울대학교 대학원

전기 · 정보공학부

전 영 한

전 영 한의 공학박사 학위논문을 인준함

2021 년 8 월

Abstract

Recently, several studies 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 analyzing the KKT condition, for deriving the backpropagation formula. In this paper, we propose a new layer formulation called the fixed-point iteration (FPI) layer that facilitates the use of more complicated operations in deep networks. The backward FPI layer is also proposed for backpropagation, which is motivated by the recurrent back-propagation (RBP) algorithm. But in contrast to RBP, the backward FPI layer yields the gradient by a small network module without an explicit calculation of the Jacobian. All 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, FPI_NN and FPI_GD, where the update operations of FPI 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 for efficient training of energy networks that have been recently studied. While RBP and its related studies have not been applied to practical examples, our experiments show the FPI layer can be successfully applied to real-world problems such as image denoising, optical flow, multi-label classification and image generation.

Keywords: Fixed-point iteration, gradient descent, differentiable layers, recurrent back-propagation, energy network, deep learning architecture

Student Number: 2014-21619

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, several papers proposed to compose a deep neural network with more complicated algorithms rather than with simple operations as it had been used. For example, there have been methods using certain types of optimization problems in deep networks such as differentiable optimization layers [5] and energy function networks [6, 12]. In these methods, the inputs 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 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 Karush–Kuhn–Tucker (KKT) conditions, etc. 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 the need to derive a separate backpropagation formula depending on the problem. Also, energy function networks can be trained efficiently by our method.

Meanwhile, there has been an algorithm called recurrent back-propagation (RBP) proposed by Almeida [4] and Pineda [51] several decades ago. RBP is a method to train an 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 by a single update equation. Even with its long history, however, RBP and related studies [9, 44] have been tested only for verifying the theoretical concept and there has been no example that applied these methods to a practical task. Moreover, there have been no studies using RBP in conjunction with other neural network components to verify the effect in more complex settings.

We take the advantages of RBP in the backpropagation process with several improvements. Our formula can be applied to general problems and combined with other network components. Unlike RBP-based studies, our method shows good performance on practical tasks.

In this paper, to facilitate the use of more complicated operations in deep networks, 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 of many numerical algorithms including most gradient-based optimizations, as a *layer* of a neural network. In the FPI layer, the layer's input and its weights 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.

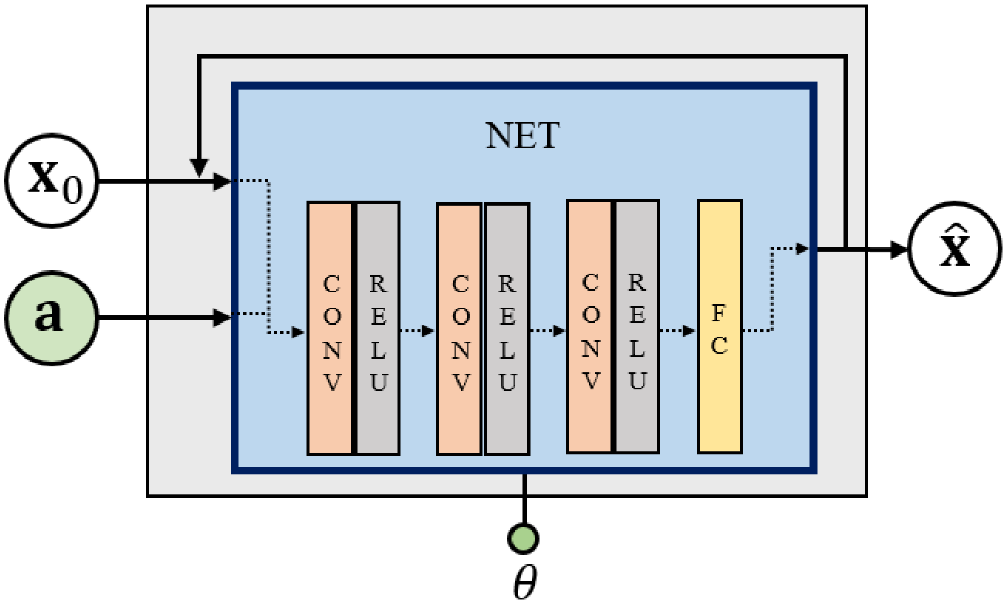


Figure 1.1: Example FPI layer.

Figure 1.1 shows the example for simple FPI layer.

We also propose a backpropagation method called *backward FPI layer* based on RBP [4, 51] to compute the derivative of the FPI layer efficiently. 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 by a small network module which allows us to avoid the explicit calculation of the Jacobian. 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 existing autograd functionalities. Especially, we provide a modularized implementation of the partial differentiation operation, which is essential in the backward FPI layer but is absent in regular autograd libraries, based on an independent computational graph. This makes the proposed method very simple to apply to various practical applications. Since FPI covers many different types of numerical algorithms as well as optimization problems, there are a lot of potential applications for the proposed method. FPI layer is highly modularized so it can be easily used together with other existing layers such as convolution, rectified linear unit (ReLU), etc., and has a richer representation power than the feedforward layer with the same number of weights. Contributions of the paper are summarized as follows.

- We propose a method to use an FPI as a layer of neural network. The FPI layer can be utilized to incorporate the mechanisms of conventional iterative algorithms, such as numerical optimization, to deep networks. Unlike other existing layers based on differentiable optimization problems, the 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 the 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 modification.
- Two types of FPI layers (FPI_NN, FPI_GD) are presented. The proposed networks based on the FPI layers are applied to practical tasks such as image denoising, optical flow, and multi-label classification, which have been largely absent in existing RBP-based studies, and show good performance.

The remainder of this paper is organized as follows: We first introduce related works in Chapter 2. Preliminary definitions and theorems are explained in Chapter 3. The proposed FPI layer is explained in Chapter 4, the experimental results follow in

Chapter 5 and 6. Finally, we conclude the paper in Chapter 7.

Chapter 2

Related Work

2.1 Energy function networks

Scalar-valued networks to estimate the energy (or error) functions have recently attracted considerable research interests. The energy function network (EFN) has a different structure from general feed-forward neural networks, and the concept was first proposed in [42]. 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) [12] performs a gradient descent on an energy function network to find the solution, and a structured support vector machine [61] loss is applied to the obtained solution. The input convex neural networks (ICNNs) [6] are defined in a specific way so 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. The deep value networks [29] and the IoU-Net [37] directly learn the loss metrics such as the intersection over union (IoU) of bounding boxes and then perform inference

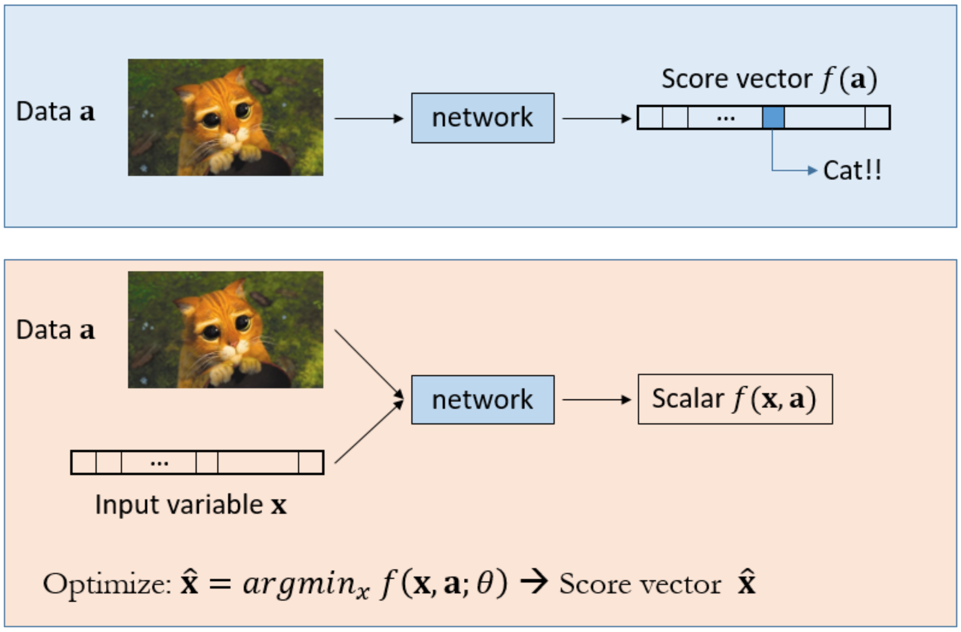

Data $\mathbf{a}$
network
Score vector $f(\mathbf{a})$
Cat!!
Data $\mathbf{a}$
network
Scalar $f(\mathbf{x}, \mathbf{a})$
Input variable $\mathbf{x}$
Optimize: $\hat{\mathbf{x}} = argmin_x\ f(\mathbf{x}, \mathbf{a}; \theta) \rightarrow$ Score vector $\hat{\mathbf{x}}$

Figure 2.1: Structure of the energy function network.

by gradient-based optimization methods. However, these methods generally require complex learning processes and each method is specialized to a limited range of applications. Moreover, they require various approximations/relaxations or constraints such as bounded conditions. On the other hand, the end-to-end SPENs [13] directly backpropagate the whole gradient-based inference process that has a fixed number of gradient steps. However, the memory requirement increases as the number of steps increases and if the number of steps is small, there is a high possibility of not converging to the optimal solution.

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 effects in more complicated problems. Moreover, they are mostly limited to a certain type of problems and require complicated learning processes. Our method can be applied to broader situations than EFN approaches, and these approaches can be equivalently implemented by the proposed method once the update equation for the optimization problem is derived.

2.2 Differentiable optimization layers

Recently, a few papers using optimization problems as a layer of a deep learning architecture have been proposed. Such a structure can contain a more complicated behavior in one layer than the usual layers in neural networks, and can potentially reduce the depth of the network. OptNet [5] 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. [2] propose 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 [22], cone program [3], semidefinite program [63], and so on. However, most of them have 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.

2.3 Recurrent back-propagation

RBP is a method to train a special case of RNN proposed by Almeida [4] and Pineda [51]. RBP computes the gradient of the steady state for an RNN with constant memory. Although RBP has great potential, it is rarely used in practical problems of deep learning. Some artificial experiments showing its memory efficiency were performed, but it was difficult to apply in complex and practical tasks. Recently, Liao et al. [44] 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 might not be sufficient for convergence in practical problems. Also, if the forward pass does not converge, the equilibrium point is meaningless so it can be unstable to train the network using the unconverged final point, which is a problem not addressed in the paper. Deep equilibrium models (DEQ) [9] tried to find the equilibrium points of a deep sequence model via an existing root-finding algorithm. Then, for back-propagation, they compute the gradient of the equilibrium point by another root-finding method. In short, both the forward and backward passes are implemented via quasi-Newton methods. DEQ can also be performed with constant memory, but it can only model the sequential (temporal) data, and the aforementioned convergence issues still exist.

RBP-based methods mainly perform experiments to verify the theoretical concepts and have not been well applied to practical examples. Our work incorporates the concept of RBP in 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 can be widely used for promotion of the RBP-based research in the deep learning field.

2.4 Learning objective functions

Using our method, we can learn the objective function without empirical knowledge. Therefore, it can be used in combination with studies in various fields that require an objective function. This section introduces the studies that use the objective function as the core.

Optimization: Mathematical optimization based on objective functions are importantly used in many fields. There have been lots of studies on optimization techniques to find an optimal value for a given objective function [23, 31, 40, 66]. Recently, several researches have emerged to find an efficient optimization strategy via machine learning [7, 43].

Inverse reinforcement learning: Studies on inverse reinforcement learning are also somewhat related to our method [1, 48, 52, 67]. The goal of reinforcement learning is to learn the policy that maximizes the sum of all rewards. On the other hand, inverse reinforcement learning aims to find a good reward function that explains a given policy well.

Optical flow: Optical flow is one of the major branches of computer vision which aims to acquire motions by matching pixels in two images. Figure 2.2 shows optical flow examples from Flying Chair dataset [24].

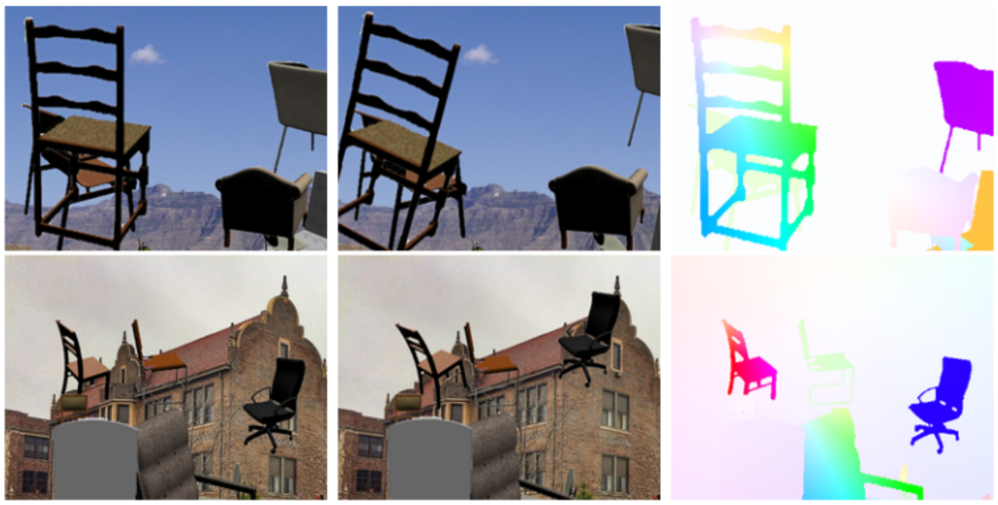


Figure 2.2: Optical flow examples.

Significant progress has been made since the study of Horn and Schunck [33], and many recent studies are based on this. They proposed an objective function consisting of a data term and a smoothness (regularization) term to solve optical flow. Since the performance was not so good at the time, various studies were conducted to improve either the objective function or the optimization method [14, 17, 49, 59, 65]. Although performance has improved considerably thanks to these studies, their modifications solve only some partial issues by heuristics and a number of hyper-parameters must be fine-tuned. Moreover, it is hard to compare the superiority between the objective functions. Meanwhile, it has not been long since the introduction of neural networks in the optical flow area. Beginning with FlowNet [24] and FlowNet2 [36], many researches have been carried out to apply neural network to optical flow [35, 60]. These methods have relatively less computational loads and they have recently improved the level of performance greatly, but designing an overall structure is very difficult and requires many empirical experiences. Unlike the traditional methods, they obtain optical flow directly from the output of the network without any energy function. Using the proposed method, optical flow can be estimated by two ways.

- Combining our method with the existing optical flow networks (using FPI_NN).
- Learning the objective function and performing optimization (using FPI_GD).

Each result is described in the Chapter **??**.

2.5 Autoencoders

Image generation is largely divided into a field based on generative adversarial network (GAN) [16, 26–28, 56] and a field based on autoencoder [32, 41]. Here we focus on the latter one.

One advantage of our method is that we can calculate the inverse function more intuitively and easily. There are various fields in deep learning that require inverse function or inverse transformation. One of the representative cases is the autoencoder [32]. Figure 2.3 shows the structure of autoencoder. The purpose of the autoencoder is to reduce the dimension, extract important information, and encode it into a latent variable

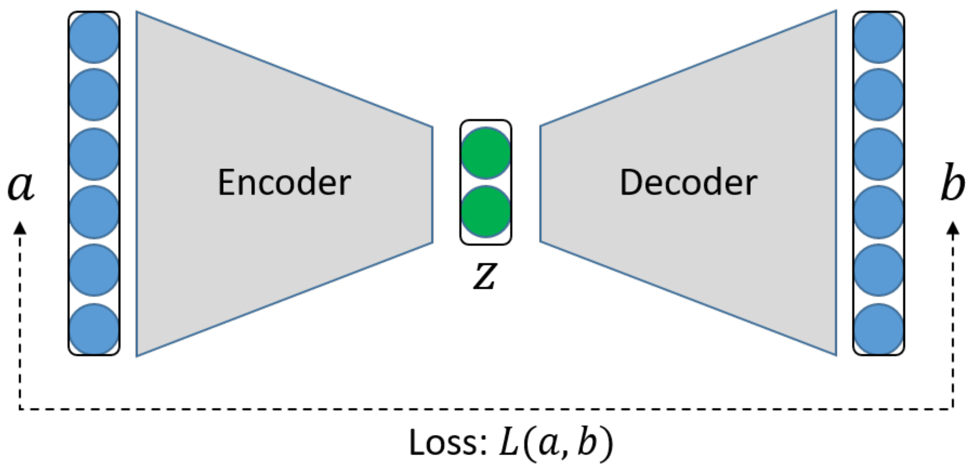


Figure 2.3: Structure of the autoencoder.

z. The decoder restores the latent variable back to high-dimensional data close to the input data. Therefore, output layer of the autoencoder has the same size as input layer and decoder is required for learning of the encoder.

Variational autoencoder (VAE) [41] has a similar structure to autoencoder, but contrary to the original autoencoder, encoder is required to train the decoder. In VAE, mean and standard deviation vectors are output of the encoder. These two vectors are combined to form a normal distribution, and latent variable **z** is created through sampling. When **z** passes through the decoder, new data similar to the existing input data can be generated. Therefore, VAE aims to generate some new data using a probability distribution and is called a generative model.

Although there are many variants [8, 15, 20, 46, 47, 53, 57, 62] of autoencoder and VAE, we show that FPI layer can effectively represent the inverse function by applying our method to the original VAE.

Chapter 3

Preliminaries

We introduce the background knowledge and preliminaries used in this dissertation.

3.1 Fixed-point iteration

For a given function g and a sequence of vectors, $\{x_n \in \mathbb{R}^d\}$, the fixed-point iteration [18] is defined by the following update equation

$$x_{n+1} = g(x_n), \ \ n = 0, 1, 2, \cdots, \tag{3.1}$$

that converges to a fixed point $\hat{x}$ of g, satisfying

$$\hat{x} = g(\hat{x}). \tag{3.2}$$

The gradient descent method ($x_{n+1} = x_n - \gamma \nabla f(x_n)$) is a popular example of fixed-point iteration. Many numerical algorithms are based on fixed-point iteration, and there are also many examples in machine learning. Here are some important concepts about fixed-point iteration.

3.2 Contraction mapping

Definition 1 (Contraction mapping) [39]. *On a metric space (X, d), the 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.*

$$d(f(x_1), f(x_2)) \leq k \cdot d(x_1, x_2). \tag{3.3}$$

Any such k is referred to as a Lipschitz constant for the function f. The smallest k that satisfies the above condition is called the (best) Lipschitz constant of f. One of the simple example of the contraction mapping is $f(x) = \frac{1}{2}x$. Here, the Lipschitz constant of f is $\frac{1}{2}$. Otherwise $f(x) = \sqrt{x}$ is not contraction mapping because $\|\sqrt{x_1} - \sqrt{x_2}\| > \|x_1 - x_2\|$ when $x_1, x_2 < \frac{1}{4}$

The distance metric is defined to be an arbitrary norm $\|\cdot\|$ in this paper. Based on the above definition, the Banach fixed-point theorem [10] states the following.

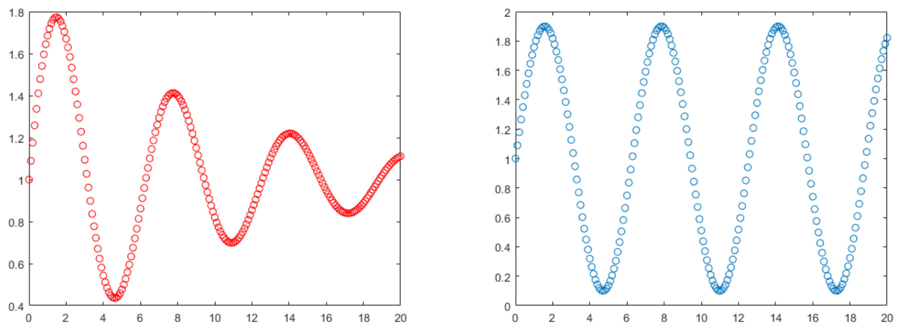
3.3 Banach fixed-point theorem

In this section, we explain the Banach fixed-point theorem (contraction mapping theorem). To this end, we first introduce several definitions which are necessary for the theorem.

Definition 2 (Cauchy sequence). *A sequence $x_0, x_1, x_2, \ldots$ on a metric space (X, d) is Cauchy sequence if for any positive real number ϵ there is a natural number N satisfies the following condition:*

$$d(x_i, x_j) < \epsilon \quad \forall i, j \geq N. \tag{3.4}$$

Figure 3.1 shows the comparison of Cauchy and non-Cauchy sequences. The Cauchy sequence converges to a point as the distance between two adjacent points gradually



(a) Cauchy (b) Not Cauchy

Figure 3.1: Example for Cauchy sequence.

decreases.

Definition 3 (Complete metric space). *A metric space (X, d) is a complete metric space if any of the following equivalent conditions are satisfied:*

- *Every Cauchy sequence of points in X has a limit that is also in X.*
- *Every Cauchy sequence in X converges in X.*
- *...*

There are many other equivalent conditions but here we only introduce conditions based on the Cauchy sequence. To explain the complete metric space in an easy to understand way, it is a metric space with no missing points in it or its boundaries. Banach fixed-point theorem is described based on the definition of the complete metric space.

Lemma 1. *For a contraction mapping f with a Lipschitz constant k and a sequence $x_{n+1} = f(x_n)$, the following inequality holds for any positive integer n:*

$$d(x_n, x_{n+1}) \leq k^n \cdot d(x_0, x_1) \tag{3.5}$$

Proof. For $n = 0$, inequality holds as follows:

$$d(x_0, x_1) \leq k^0 \cdot d(x_0, x_1). \tag{3.6}$$

Let assume the inequality holds when $n = m$:

$$d(x_m, x_{m+1}) \leq k^m \cdot d(x_0, x_1) \tag{3.7}$$

Then by the definition of contraction mapping,

$$\begin{aligned} d(x_{m+1}, x_{m+2}) &= d(f(x_m), f(x_{m+1})) \\ &\leq k \cdot d(x_m, x_{m+1}) \\ &\leq k \cdot k^m \cdot d(x_0, x_1) \\ &= k^{m+1} \cdot d(x_0, x_1), \end{aligned} \tag{3.8}$$

which implies that the inequality holds when $n = m + 1$. By the mathematical induction, the inequality holds for all positive integer n. □

Using this lemma, the following Banach fixed-point theorem is proved.

Theorem 1 (Banach fixed-point theorem) [10]. *A contraction mapping $f : X \to X$ on a complete metric space (X, d) has exactly one fixed point and it can be found by starting with any initial point and iterating the update equation until convergence.*

Proof. By the Lemma 1,

$$d(x_n, x_{n+1}) \leq k^n \cdot d(x_0, x_1). \tag{3.9}$$

For any positive integer i, j that satisfies $i < j$,

$$\begin{aligned}
d(x_i, x_j) &\leq \sum_{n=i}^{j-1} d(x_n, x_{n+1}) \\
&\leq \sum_{n=i}^{j-1} k^n \cdot d(x_0, x_1) \\
&= d(x_0, x_1) \cdot \sum_{n=i}^{j-1} k^n \\
&\leq d(x_0, x_1) \cdot \sum_{n=i}^{\infty} k^n \\
&= d(x_0, x_1) \cdot \frac{k^i}{1-k}.
\end{aligned} \tag{3.10}$$

For any positive real number ϵ we can find a large number N that satisfies

$$k^i < \frac{(1-k)\epsilon}{d(x_0, x_1)}. \tag{3.11}$$

If we choose $i, j > N$,

$$\begin{aligned}
d(x_i, x_j) &\leq d(x_0, x_1) \cdot \frac{k^i}{1-k} \\
&\leq d(x_0, x_1) \cdot \frac{1}{1-k} \cdot \frac{(1-k)\epsilon}{d(x_0, x_1)} \\
&= \epsilon.
\end{aligned} \tag{3.12}$$

So $\{x_n\}$ is a Cauchy sequence and the limit $\hat{x}$ is fixed point and $\hat{x}$ is in X by completeness:

$$\begin{aligned}
\hat{x} &= \lim_{n\to\infty} x_n \\
&= \lim_{n\to\infty} f(x_{n-1}) \\
&= f(\lim_{n\to\infty} x_{n-1}) \\
&= f(\hat{x}).
\end{aligned} \tag{3.13}$$

By the definition of contraction mapping, f has only one fixed point by contradiction:

$$\begin{aligned} d(f(\hat{x}), f(\hat{x}')) &= d(\hat{x}, \hat{x}') \\ &> k \cdot d(\hat{x}, \hat{x}') \end{aligned} \tag{3.14}$$

□

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.

3.4 Contraction property

Here we introduce the known contraction property for the Jacobian $J_g(x)$ of the function $g(x)$.

Theorem 2. *For the convex set A and the function $g : A \to \mathbb{R}^n$, if the matrix norm of Jacobian satisfies*

$$\|J_g(\hat{x})\| \ \leq k < 1, \tag{3.15}$$

g is a contraction mapping.

Proof. Let $G(t) = g(x + t(y - x))$ for $t \in [0, 1]$. Then we have

$$\begin{aligned} g(y) - g(x) &= G(1) - G(0) \\ &= \int_0^1 G'(t)\, dt \\ &= \int_0^1 J_g(x + t(y - x))(y - x)\, dt. \end{aligned} \tag{3.16}$$

By the triangle inequality,

$$
\begin{aligned}
\|g(y) - g(x)\| &\leq \int_0^1 \|J_g(x + t(y - x))(y - x)\| \, dt \\
&\leq \int_0^1 \|J_g(x + t(y - x))\| \cdot \|y - x\| \, dt \\
&\leq k \, \|y - x\|.
\end{aligned} \tag{3.17}
$$

By the definition of the contraction mapping, g is a contraction mapping.

□

We prove the inverse of Theorem 2 in Section 4.4 which is included in Proposition 1. (If g is a contraction mapping, the matrix norm of Jacobian is less than $k < 1$).

Chapter 4

Proposed Method

The fixed-point iteration formula contains a wide variety of forms and can be applied to most iterative algorithms. Section 4.1 describes the basic structure and principles of the FPI layer. Section 4.2 and 4.3 explains the differentiation of the layer for backpropagation. Section 4.4 describes the convergence of the FPI layer. Section 4.5 presents two exemplar cases of the FPI layer.

4.1 Structure of the FPI 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 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:

$$x_{n+1} = g(x_n, z; \theta), \tag{4.1}$$

$$\hat{x} = \lim_{n\to\infty} x_n \tag{4.2}$$

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

$$\begin{aligned}
\hat{x} &= g(\hat{x}, z; \theta) \\
&= \mathbf{F}(x_0, z; \theta) \\
&= \lim_{n\to\infty} g^{(n)}(x_0, z; \theta) \\
&= (g \circ g \circ \cdots \circ g)(x_0, z; \theta)
\end{aligned} \tag{4.3}$$

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 predetermined to any value such as zero matrix. 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, the first equation in (4.1) is repeated until convergence to find the output $\hat{x}$. We may use some acceleration techniques such as the Anderson acceleration [50] for faster convergence.

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 do not have to be same.

4.2 Differentiation of the FPI layer

Similar to 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 node in the computational graph.Note that

$$\hat{x} = g(\hat{x}, z; \theta). \tag{4.4}$$

is satisfied at the fixed point $\hat{x}$. If we differentiate both sides of the above equation w.r.t. θ, we have

$$\frac{\partial \hat{x}}{\partial \theta} = \frac{\partial g}{\partial \theta}(\hat{x}, z; \theta) + \frac{\partial g}{\partial x}(\hat{x}, z; \theta)\, \frac{\partial \hat{x}}{\partial \theta}. \tag{4.5}$$

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

$$\frac{\partial \hat{x}}{\partial \theta} = \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{-1} \frac{\partial g}{\partial \theta}(\hat{x}, z; \theta), \tag{4.6}$$

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 $\hat{x}$. One downside of the above derivation is that it requires the calculation of Jacobians of g, which may need a lot of memory space (e.g., convolution layers). Moreover, calculating the inverse can also be a burden. In the next section, we will provide an efficient way to resolve these issues.

4.3 Backward FPI layer

To train the FPI layer, we need to obtain the gradient w.r.t. its parameter θ. In contrast to RBP [4, 51], we propose a computationally efficient layer, called the backward

FPI layer, that yields the gradient without explicitly calculating the Jacobian. Here, we assume that an 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 (4.6), we have

$$\begin{aligned} \nabla_\theta L &= \left(\frac{\partial \hat{x}}{\partial \theta} \right)^\top \nabla_{\hat{x}} L \\ &= \left(\frac{\partial g}{\partial \theta}(\hat{x}, z; \theta) \right)^\top \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{-\top} \nabla_{\hat{x}} L. \end{aligned} \tag{4.7}$$

This section describes how to calculate the above equation efficiently. (4.7) can be divided into two steps as follows:

$$c = \left(I - \frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^{-\top} \nabla_{\hat{x}} L, \tag{4.8}$$

$$\nabla_\theta L = \left(\frac{\partial g}{\partial \theta}(\hat{x}, z; \theta) \right)^\top c. \tag{4.9}$$

Rearranging (4.8) 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.,

$$c_{n+1} = \left(\frac{\partial g}{\partial x}(\hat{x}, z; \theta) \right)^\top c_n + \nabla_{\hat{x}} L, \tag{4.10}$$

which corresponds to RBP. This iteration eliminates the need of 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, i.e. the backward FPI layer, that yields the gradient without an explicit calculation of the Jacobian. To this end, we define a new function h as

$$h(x, z, c; \theta) = c^\top g(x, z; \theta), \tag{4.11}$$

then (4.10) becomes

$$c_{n+1} = \frac{\partial h}{\partial x}(\hat{x}, z, c_n; \theta) + \nabla_{\hat{x}} L. \tag{4.12}$$

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, (4.9) is expressed using h:

$$\nabla_\theta L = \frac{\partial h}{\partial \theta}(\hat{x}, z, c; \theta), \tag{4.13}$$

where c is the fixed point obtained from the fixed-point iteration in (4.12). In this way, we can compute $\nabla_\theta L$ by (4.13) without any memory-intensive operations and Jacobian calculation. 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 layer 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.

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 about the above derivation in that the differentiations w.r.t. x and θ are *partial* differentiations. x and θ might have some dependency with each other, which can disrupt the partial differentiation process if it is computed based on a usual autograd framework. Let $\phi(a, b)$ hereafter denotes 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 will also be a set of derivatives.

In order to resolve the issue, we implemented a general partial differentiation operator

$$P(s; r) \triangleq \frac{\partial r(s)}{\partial s} \tag{4.14}$$

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 $I(t)$ denotes 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 to each other. Here, the resulting derivatives $\partial r'/\partial s'$ are also attached in 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

$$\eta = H(\delta; \rho) \tag{4.15}$$

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, and a plus operator can construct (4.12) and (4.13) entirely, and the iteration of (4.12) 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 Appendix section. All the forward and backward layers are implemented at a high level of abstraction, and therefore, it can be easily applied to practical tasks by changing the structure of g to one that is suitable for each task.

4.4 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. In practice, we empirically found out that setting the initial values of weights (θ) to small values is enough for making 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* (4.10) *converges to a unique point.*

Proof. For simplicity, we omit z and θ from g. By the definition of the contraction mapping and the assumption of the arbitrary norm metric,

$$\frac{\|g(x_2) - g(x_1)\|}{\|x_2 - x_1\|} \leq k \tag{4.16}$$

is satisfied for all x_1 and x_2 ($0 \leq k < 1$). For a unit vector v, i.e., $\|v\| = 1$ for the aforementioned norm, and a scalar t, let $x_2 = x_1 + tv$. Then, the above inequality becomes

$$\frac{\|g(x_1 + tv) - g(x_1)\|}{\|t\|} \leq k. \tag{4.17}$$

For another vector u with $\|u\|_*\leq 1$ where $\|\cdot\|_*$ indicates the dual norm, it satisfies

$$\begin{aligned} \frac{u^\top(g(x_1 + tv) - g(x_1))}{|t|} &\leq \frac{\|g(x_1 + tv) - g(x_1))\|}{|t|} \\ &\leq k \end{aligned} \tag{4.18}$$

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

$$\begin{aligned} \lim_{t \to 0^+} \frac{u^\top(g(x_1 + tv) - g(x_1))}{|t|} &= \nabla_v(u^\top g)(x_1) \\ &\leq k. \end{aligned} \tag{4.19}$$

According to the chain rule,

$$\nabla(u^\top g) = (u^\top J_g)^\top \tag{4.20}$$

where J_g is the Jacobian of g. That gives

$$\begin{aligned} \nabla_v(u^\top g)(x_1) &= (\nabla(u^\top g)(x_1))^\top \cdot v \\ &= u^\top J_g(x_1)\, v \\ &\leq k. \end{aligned} \tag{4.21}$$

Let $x_1 = \hat{x}$ then

$$u^\top J_g(\hat{x})\, v \leq k \tag{4.22}$$

for all u, v that satisfy

$$\begin{aligned} \|u\|_* &\leq 1, \\ \|v\| &= 1. \end{aligned} \tag{4.23}$$

Therefore,

$$\begin{aligned} \|J_g(\hat{x})\| &= \sup_{\|v\|=1} \|J_g(\hat{x})v\| \\ &= \sup_{\|v\|=1, \|u\|_* \le 1} u^\top J_g(\hat{x})v \\ &\le k < 1 \end{aligned} \tag{4.24}$$

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. □

4.5 Two representative cases of the FPI layer

As mentioned before, FPI can take a wide variety of forms. We present two representative methods that are easy to apply to practical problems.

4.5.1 Neural net FPI layer (FPI_NN)

The most intuitive way to use the FPI layer is to set g to 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. Algorithm 1 shows the process of single FPI_NN network. a_i is a training sample and p_i is its corresponding ground truth (answer).

Convergence threshold τ can be differ for forward and backward layer. Note that the same fixed-point iteration module is used for both forward and backward FPI. The performance of our algorithm is most affected by the τ value as there are few hyperparameters. Smaller τ value usually results in better performance, but takes a lot of time (especially for learning). In experiments, the maximum number of iterations is set in order to prevent the code from taking too much time in real experiments. FPI_NN

Algorithm 1: Single FPI_NN network

```
Input: Training set $S = \{(a_i, p_i)|1 \leq i \leq N\}$, mini-batch size $b$,
  convergence criterion $\beta$, convergence threshold $\tau$
Initialize: Parameters $\theta$ of neural network $f$
for number of training epochs do
    Initialize $\bar{B} = \{B|B$ is a partition of $S$ with size $b\}$.
    for $B \in \bar{B}$ do
        for $(a_j, p_j) \in B$ do
            Initialize $x_0, \quad n = 0$.
            while $\beta(x_n, x_{n+1}) < \tau$ do
                $g(x, a_j; \theta) = f(x, a_j; \theta)$.
                $x_{n+1} = g(x_n, a_j; \theta), \quad n = n + 1$.
            end
            $\hat{x}_j = x_n$,   Initialize $c_0, \quad n = 0$.
            while $\beta(c_n, c_{n+1}) < \tau$ do
                $h(x, a_j, c; \theta) = c^\top g(x, a_j; \theta)$.
                $c_{n+1} = \frac{\partial h}{\partial x}(\hat{x}_j, a_j, c_n; \theta) + \nabla_{\hat{x}_j} L, \quad n = n + 1$.
            end
            $\hat{c}_j = c_n, \quad \nabla_\theta L_j = \frac{\partial h}{\partial \theta}(\hat{x}_j, a_j, \hat{c}_j; \theta)$
        end
        Update $\theta$ by $\frac{1}{b} \sum_{(a_j, p_j) \in B} \nabla_\theta L_j$
    end
end
```

can perform more complicated behaviors with the same number of parameters than using g directly without FPI, as demonstrated in the experiments section.

4.5.2 Gradient descent FPI layer (FPI_GD)

The gradient descent method can be a perfect example for the FPI layer. This can be used for efficient implementations of the energy function networks like ICNN [6]. Energy function networks are scalar-valued networks for estimating the energy (or error) functions. Unlike a typical network which obtains the answer directly as the output of the network (i.e., $f(a;\theta)$ is the answer of the query a), an energy function network retrieves the answer by optimizing an input variable of the network (i.e., $\arg\min_x f(x,a;\theta)$ becomes the answer). The easiest way to optimize the network f is gradient descent

$$x_{n+1} = x_n - \gamma \nabla f(x_n, a; \theta). \tag{4.25}$$

This is a form of FPI and the fixed point $\hat{x}$ is the local minimum of f, i.e.,

$$\nabla f(\hat{x}, a; \theta) = 0. \tag{4.26}$$

If f is convex, $\hat{x}$ becomes the global optimum of $f(x)$.

$$\hat{x} = \arg\min_x f(x, a; \theta). \tag{4.27}$$

We can enforce f to be a convex function using special structure like [6]. However in the experiments, it has better performance to use the local minimum.

In case of a single FPI layer network with FPI_GD, $\hat{x}$ 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 as follows:

$$\min_\theta L(x^*, \hat{x}). \tag{4.28}$$

This behavior conforms to that of an energy function network. However, unlike the existing methods, the proposed method can be trained easily with the universal back-propagation formula. Therefore, the proposed FPI layer can be an effective alternative to train energy function networks.

Algorithm 2 shows the process of single FPI_GD network. An advantage of FPI_GD is that it can easily satisfy the bounded Lipschitz condition by adjusting the step size γ. In other words, it does not matter how the network that creates the objective function is configured.

Algorithm 2: Single FPI_GD network

```
1  Input: Training set $S = \{(a_i, p_i) | 1 \leq i \leq N\}$, mini-batch size $b$,
   convergence criterion $\beta$, convergence threshold $\tau$, step size $\gamma$
2  Initialize: Parameters $\theta$ of energy function network $f$
3  for number of training epochs do
4      Initialize $\bar{B} = \{B | B$ is a partition of $S$ with size $b\}$.
5      for $B \in \bar{B}$ do
6          for $(a_j, p_j) \in B$ do
7              Initialize $x_0, \quad n = 0$.
8              while $\beta(x_n, x_{n+1}) < \tau$ do
9                  $g(x, a_j; \theta) = x - \gamma \nabla f(x, a_j; \theta)$.
10                 $x_{n+1} = g(x_n, a_j; \theta), \quad n = n + 1$.
11             end
12             $\hat{x}_j = x_n$, Initialize $c_0, \quad n = 0$.
13             while $\beta(c_n, c_{n+1}) < \tau$ do
14                 $h(x, a_j, c; \theta) = c^\top g(x, a_j; \theta)$.
15                 $c_{n+1} = \frac{\partial h}{\partial x}(\hat{x}_j, a_j, c_n; \theta) + \nabla_{\hat{x}_j} L, \quad n = n + 1$.
16             end
17             $\hat{c}_j = c_n, \quad \nabla_\theta L_j = \frac{\partial h}{\partial \theta}(\hat{x}_j, a_j, \hat{c}_j; \theta)$
18         end
19         Update $\theta$ by $\frac{1}{b} \sum_{(a_j, p_j) \in B} \nabla_\theta L_j$
20     end
21 end
```

Chapter 5

Applications to optimization and classification tasks

Since several studies [4, 9, 44, 51] have already shown that RBP-based algorithms require only a constant amount of memory, we omit memory-related experiments. Instead, we focus 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 alteration. The only difference was the choice of g where we could simply plug in its functional definition, which shows the efficiency of the proposed framework. For all experiments, the detailed structure of g and the hyperparameters for training are described in the Appendix section. Although the performance is not recorded in this section, we can also use Anderson [50] acceleration for fast convergence although it has a slight performance penalty. We used a small number of layers when constructing g in all experiments. This is because even if the number of layers is different, if the total number of parameters is the same, the performance is almost the same. All training was performed using the Adam [40] optimizer.

5.1 Toy examples: optimization problems

Here, we show the feasibility of our algorithm by learning optimization problems. 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.

5.1.1 Training objective functions by FPI_GD

FPI_GD method learns the form of the objective function in the training phase so that optimizing the objective function for each sample input $\mathbf{a}$ yields $\mathbf{x}^* \approx \mathbf{t}$. Accordingly, in the test phase, we optimize the trained objective function for a new sample input and compare the result to the ground truth target. In other words, this problem can be seen as finding an unconstrained objective function that gives a similar answer to the given constrained problem. Of course, the more complex the problem, the more difficult it is to find the objective function. However it learns the objective function quite accurately about the four optimization problems in this section. We used single FPI layer networks for this problem.

5.1.1.1 Translation

This is the simplest toy example where the desired solution for a sample input $\mathbf{a}$ is $\mathbf{t} = \mathbf{a} + \mathbf{d}$ for some vector $\mathbf{d}$. $\mathbf{d}$ is randomly picked and fixed for the whole process. One of ideal objective functions for this problem is $||\mathbf{x} - (\mathbf{a} + \mathbf{d})||^2$.

5.1.1.2 Linear equation

Here we consider the case of linear equation $W\mathbf{x} = \mathbf{a}$ where W is a 10×10 matrix with full rank. It can be considered as a convex optimization problem:

$$\underset{\mathbf{x}}{\text{minimize}} \qquad ||W\mathbf{x} - \mathbf{a}||^2 \,. \tag{5.1}$$

The desired value should be $\mathbf{t} = W^{-1}\mathbf{a}$.

5.1.1.3 Constrained problem 1. Box constraint

This is a box-constrained problem:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \|\mathbf{x}-\mathbf{a}\|^2, \\ \text{subject to} \quad & -\mathbf{1} \le \mathbf{x} \le \mathbf{1}. \end{aligned} \tag{5.2}$$

Here, the inequalities are element-wise. This can be reformulated as an unconstrained problem with an extended-real-valued function $\|\mathbf{x}-\mathbf{a}\|^2 + I_{\text{box}}(\mathbf{x})$, where $I_{\text{box}}(\mathbf{x})$ is zero for $-\mathbf{1} \le \mathbf{x} \le \mathbf{1}$ and infinity otherwise. Target $\mathbf{t}$ can be found by clipping the elements of $\mathbf{a}$ into the range $[-1, 1]$.

5.1.1.4 Constrained problem 2. Standard simplex

This is another constrained problem that finds the closest discrete probability distribution from a given vector:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \|\mathbf{x}-\mathbf{a}\|^2, \\ \text{subject to} \quad & \mathbf{1}^T\mathbf{x} = 1, \\ & \mathbf{x} \ge \mathbf{0}. \end{aligned} \tag{5.3}$$

Likewise, this can be reformulated as an unconstrained problem with $\|\mathbf{x}-\mathbf{a}\|^2 + I_{\text{simp}}(\mathbf{x})$ where $I_{\text{simp}}(\mathbf{x})$ is analogously defined as in the previous problem. The desired solution can be calculated based on [64].

All the toy examples were evaluated under the following settings: Training and test sample inputs were randomly generated by zero-mean Gaussian distribution with standard deviation 2 and 1, respectively. For training, 10,000 random samples were generated in every epoch. For test, 1,000 samples were generated and fixed for the entire procedure. The dimensions of $\mathbf{x}$, $\mathbf{a}$, and $\mathbf{t}$ were 10, and the size of mini-batch was 100. For all problems, mean squared error (MSE) was used as the loss metric L.

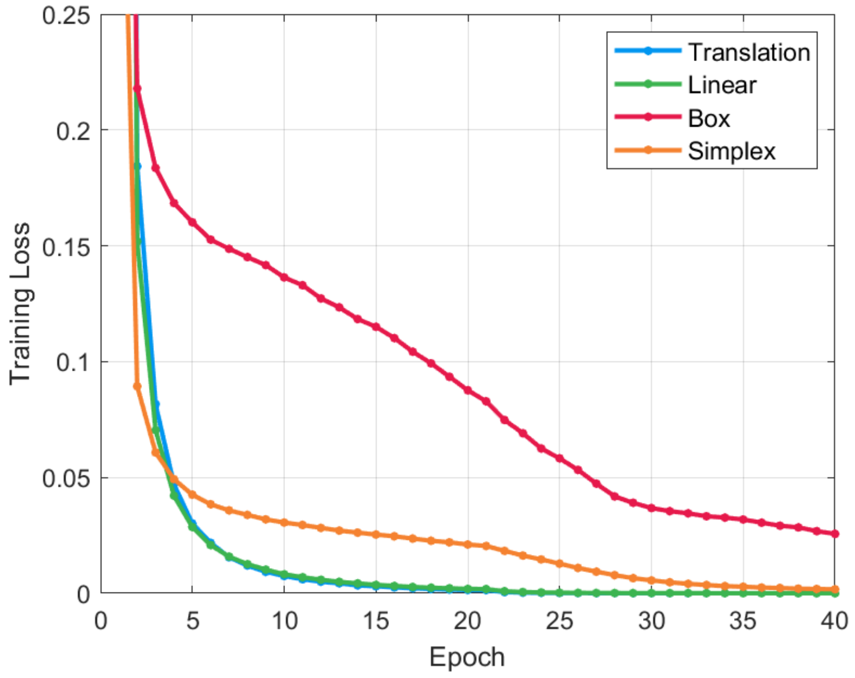


Figure 5.1: Training loss per epoch in toy examples.

The step size γ was set to 10^3 and the number of gradient steps N to 1,000. We used a simple two-layer network, i.e.,

[vector input] - (20×64 FC) - (ReLU) - (64×64 FC) - (MS) - [scalar output].

Here, the input is the concatenation of $\mathbf{x}$ and $\mathbf{a}$, and FC and MS represent the fully-connected and the mean square operations, respectively. This structure was applied to every toy example and the Adam [40] optimizer was used in the outer process.

Figure 5.1-5.4 shows the experimental results for the toy examples. Here, we can confirm that the losses of each problem decrease nicely along the training epochs as well as the gradient descent steps. The shape of a learned objective function was drawn by perturbing the variable input near the target for a fixed sample input. We randomly picked 10 directions for perturbation and displayed the average and standard deviation

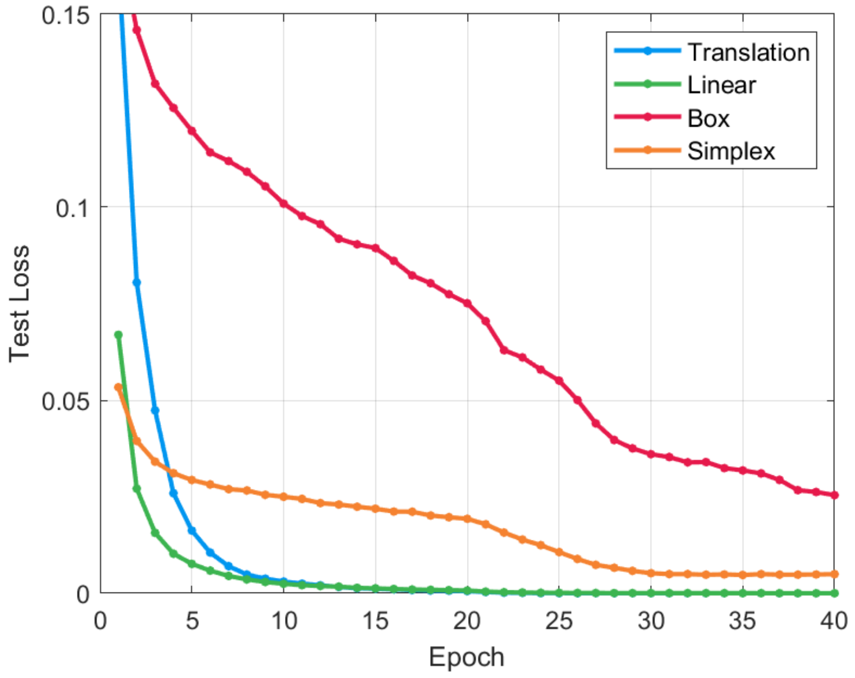


Figure 5.2: Test loss per epoch in toy examples.

of the objective function for each length of perturbation. Here, the figure shows that the learned objective functions have ideal shapes for optimization.

5.1.2 Additional experiment on optimization problem

Here, we show the feasibility for both of the FPI_GD and FPI_NN by learning a constrained optimization problem with a box constraint. The performance was evaluated for the FPI_NN network, the FPI_GD network, and a non-FPI network which has the same structure with g of FPI_NN. The structures of g of FPI_NN and the energy function of FPI_GD were both linear-ReLU-linear. The dimension of a, x, and the number of hidden nodes were 10, 10, and 32, respectively. We randomly generated a from zero-mean Gaussian distributions. 10,000 training samples were generated with $\sigma = 2$

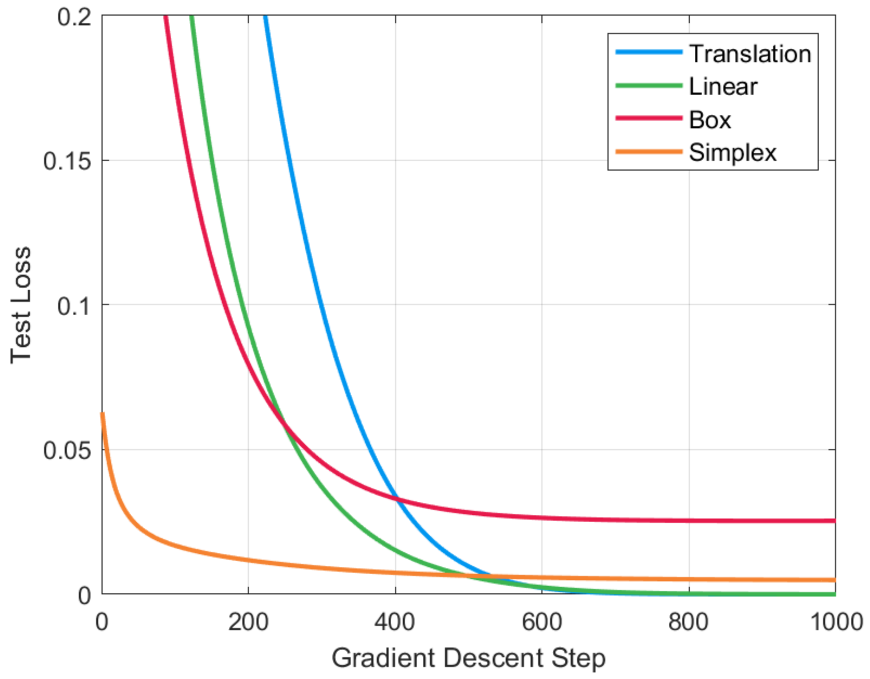


Figure 5.3: Test loss per gradient descent step.

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 FPI_GD was fixed to 0.01. All the models were trained for 40 epochs with batch size 100, and the training and test losses (MSE) were reported.

Figure 5.5 shows the train and test losses per epoch. Here, we can see that FPI_NN outperforms the other networks for both the train and the test losses.

5.2 Multi-label classification

The multi-label text classification dataset (Bibtex) was introduced in [38]. The goal of the task is to find the correlation between the data and the multi-label features. Both the data and features are binary with 1836 indicators and 159 labels, respectively. The

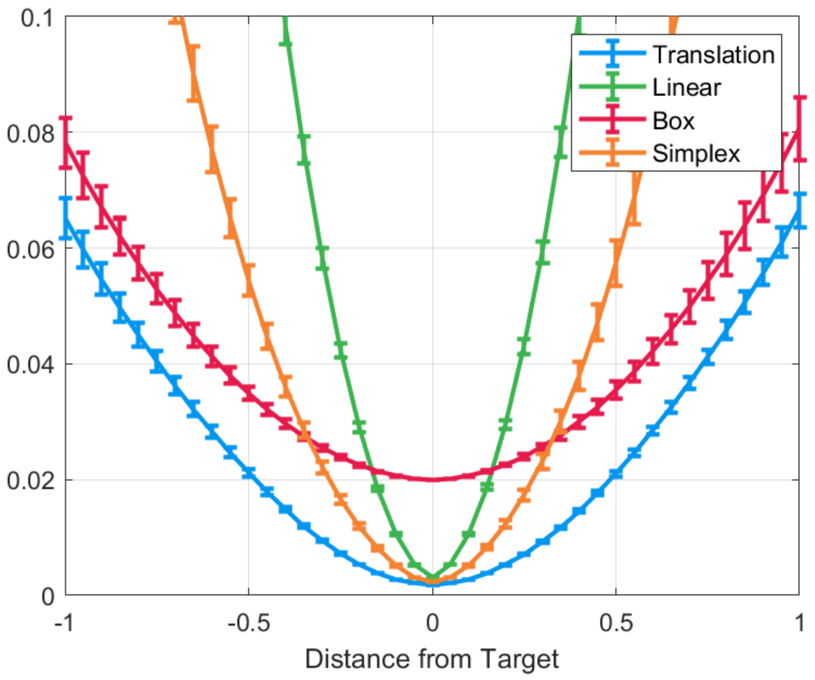


Figure 5.4: Objective function.

numbers of indicators and labels differ for each data, and that of labels is unknown during the evaluation process. We used the same train and test split as in [38] and evaluated the F_1 scores. Here, we used two single FPI layer networks with FPI_GD and FPI_NN. We set g of FPI_NN and f of FPI_GD to similar structures which contain one or two fully-connected layers and activation functions. As mentioned, the detailed structures of the networks are described in Appendix section.

Table 5.1 shows the F_1 scores. GT stands for ground truth. Here, DVN(Adversarial) achieves the best performance, however, it generates adversarial samples for data augmentation. Both FPI_GD layer and FPI_NN layer achieves better performance than DVN(GT) under the same condition. Despite their simple structures, our algorithms perform the best among those using only the training data, which confirms the effec-

Method	F1 score
MLP [12]	38.9
Feedforward net [6]	39.6
SPEN [12]	42.2
ICNN [6]	41.5
DVN(GT) [11]	42.9
DVN(Adversarial)[29]	**44.7**
FPI_GD layer (Ours)	43.2
FPI_NN layer (Ours)	43.4

Table 5.1: F_1 score of multi-label text classification (higher is better). Our method shows the best performance among those using only the training data.

tiveness of the proposed method.

5.3 Implementation details

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

$$\beta = \frac{||x_{n+1} - x_n||^2}{||x_n||^2} \tag{5.4}$$

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

1. **FPI_NN layer:** To see the full potential of the FPI layer, we tested an FPI layer with g being a general (small) neural network module. In this case, g can be-

come 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. In practice, using small initial weights for g was sufficient in our empirical experience.

2. **FPI_GD layer:** Inspired by the energy function networks [6, 12, 29], 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 energy function networks, our version can perform backpropagation easily with the existing autograd functionalities.

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 is passed onto g, e.g., for vector inputs

$$g(x, z; \theta) = g([x^T \ z^T]^T; \theta). \tag{5.5}$$

Accordingly, the size of the input of g was bigger than that of the output. x_0 was either a zero vector or a zero matrix in all the experiments.

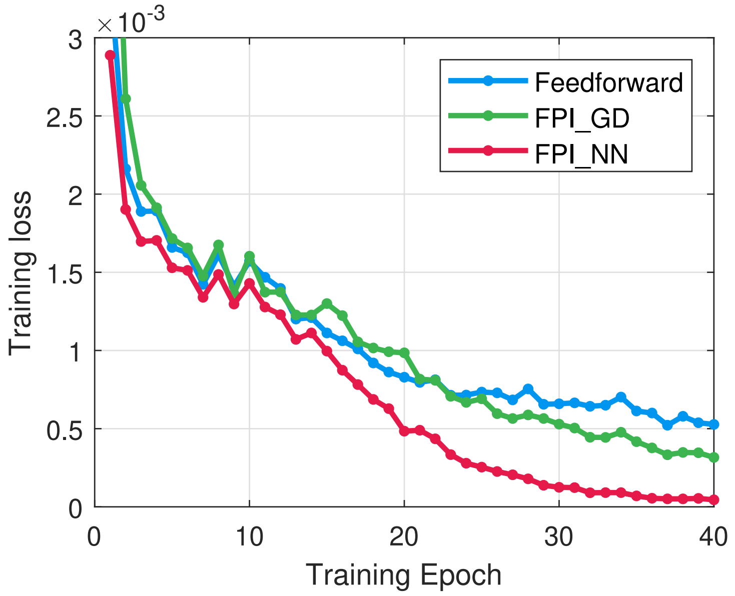
All the training was performed using the Adam optimizer with learning rate 10^{-3}, and no weight decay was used. In the following experiments, we used the ReLU activation function most of the time. Although this does not exactly align with the assumption in the paper, we used it anyway as in many practices of deep learning and confirmed that the FPI layer still performs nicely.

5.3.1 Multi-label classification

For this experiment, all the training settings of the proposed methods were the same as the other compared algorithms, except for the network structure. The network structure of 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 have a scalar output. Here, the number of hidden nodes were also 512, and an additional sigmoid layer was added after the FPI_GD layer. We fixed the step size to 1.0 for the gradient descent in FPI_GD and used a different convergence criterion as follows:

$$\beta' = \|x_{n+1} - x_n\|^2 \tag{5.6}$$

where the convergence threshold was 10^{-4}. The sizes of both the networks' inputs and outputs were 1836 and 159, respectively, which is the same as the numbers of indicators and labels.



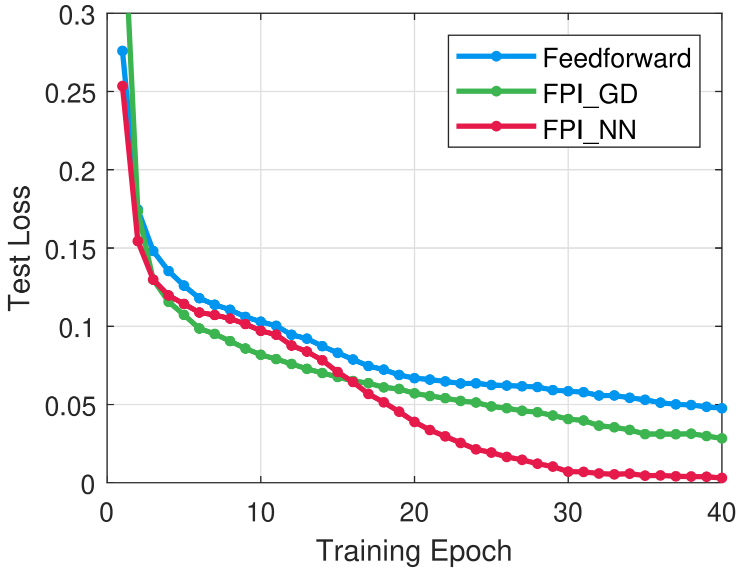


Figure 5.5: Training and test loss per epoch in constrained problem. The FPI_GD and FPI_NN networks perform better than the feedforward network with the same number of parameters.

Chapter 6

Applications to computer vision tasks

6.1 Image denoising

Here, we compare the image denoising performance for gray images perturbed by Gaussian noise with variance σ^2. Image denoising has been traditionally solved with iterative, numerical algorithms, hence using an iterative structure like 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 [24] and converted it to gray scale (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 6.1.

Table 6.1 shows that the single FPI layer network outperforms the feedforward network in all experiments.

The performance gap between the two algorithms is larger in more noisy circumstances as shown in Table 6.2. Since both the networks are trained to yield the best

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

Table 6.1: Denoising performance (PSNR, higher is better). The single FPI_NN network outperforms the feedforward network.

	$\sigma = 15$	$\sigma = 20$	$\sigma = 25$
PSNR gap	0.25	0.56	0.65

Table 6.2: Performance gap between feedforward and FPI_NN method. The gap is larger for noisier (difficult) circumstances.

performance in their given settings, this confirms that a structure with repeated operations can be more suitable for this type of problem. An 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 was no specialized component except the bare definition of g thanks to the highly modularized nature of the layer.

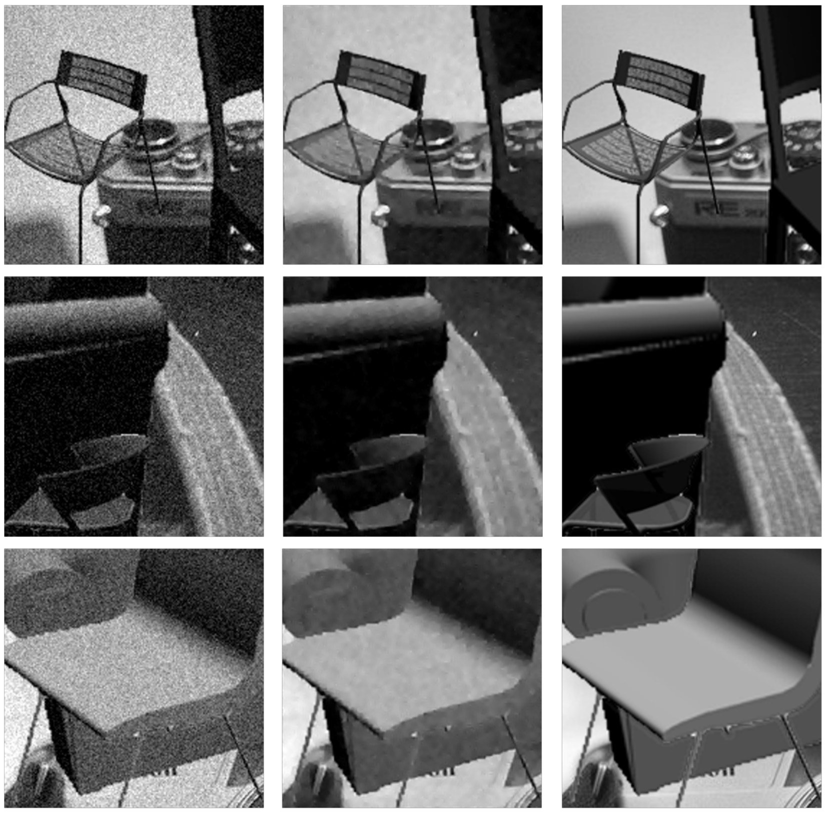


Figure 6.1: Image denoising examples. The left column shows the noisy input and the right column is the ground truth. The denoised images are shown in the middle.

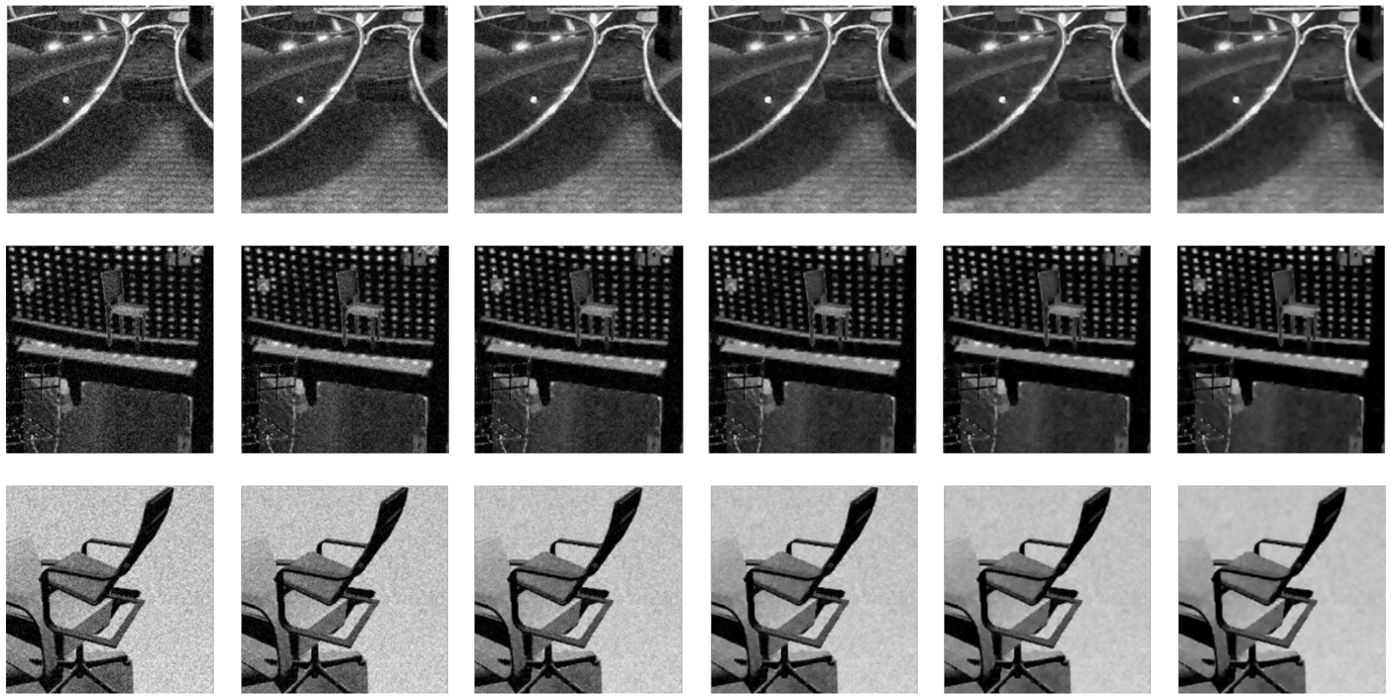


Figure 6.2: Image denoising examples corresponding to the number of iterations. Last column shows the final result.

Figures 6.1 and 6.2 show the image denoising examples.

6.2 Optical flow

6.2.1 Estimation by neural network

Optical flow is one of the major research areas of computer vision that aims to acquire motions by matching pixels in two images. First, we demonstrate the effectiveness of the FPI layer by a simple experiment, where the layer is attached at the end of FlowNet [24]. The Flying Chairs dataset [24] was used with the original split which has 22,232 training and 640 test samples. Figure 6.3 shows example images for Flying Chair dataset. 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 6.4. Although the number of additional parameters is extremely small (less than 0.01%) and the computation time is nearly the same with the original FlowNet, it shows noticeable performance improvement.

6.2.2 Estimation by objective function

There have been suggested a lot of objective functions to find optical flow [17, 33, 55, 65]. Many of them are based on the widely-used Horn-Schunck (HS) energy function [33]. The usual implementation of the HS method is based on an approximated version of the original objective function, i.e.,

$$\begin{aligned} &E(I_1, I_2, u, v) = \\ &\iint \left[(I_1(x,\ y) - I_2(x+u,\ y+v))^2 + \lambda \left(\|\nabla u\|^2 + \|\nabla v\|^2 \right) \right] \, \mathrm{d}x\mathrm{d}y, \end{aligned} \tag{6.1}$$

where x and y are horizontal and vertical coordinates, respectively, I_1 and I_2 are the two images, and (u, v) is an optical flow vector at pixel (x, y). The first term

$(I_1(x,\ y) - I_2(x+u,\ y+v))^2$ represents the photometric consistency and the second term implies the similarity of flow vectors between nearby pixels. This manually designed energy function has been modified by numerous studies to overcome its poor performance. However, many of these modifications were designed heuristically and it is difficult to know which is exactly better. The motivation of this experiment is what if we learn the objective function itself instead of manually designing it. In most cases, objective functions are handcrafted with some unrealistic assumptions, heuristics, and approximations, and they are constantly modified by trial-and-error. Thus, it is hard to validate the effectiveness of a given design and is also difficult to compare different choices, because in most cases it requires a fair amount of actual evaluations. Moreover, there are usually a number of hyper-parameters such as λ in equation (6.1) that must be fine-tuned, so it is difficult to guarantee consistent performance in various environments.

It is well-known that it is possible for neural networks to approximate variety of functions, according to the universal approximation theorem [21, 34], if we have enough resources such as computation power, memory, and training data. However, in reality, resources are limited and it requires some efforts to apply the proposed method efficiently in practical applications. Most importantly, one has to consider the characteristics of the application and choose an adequate structure to reduce the search space. For example, one can design a network structure that is similar to, but more general than, an existing handcrafted objective function for a particular application.

To apply the proposed method to optical flow, we design a neural network for the inner objective utilizing the basic form of the HS energy function (6.1), instead of using a black-box neural network. Let us assume that $J \triangleq f_e(I;\theta)$ is a feature extraction network for images, where its result J is an image with the same width and height as I, but with different channels. Then the inner objective for optical flow is defined as follows:

$$\begin{aligned} f_{\mathrm{op}}(u, v, I_1, I_2; \theta) &\triangleq E(u, v, J_1, J_2) = \\ &\iint \left[(J_1(x,\ y) - J_2(x+u,\ y+v))^2 + \lambda \left(\|\nabla u\|^2 + \|\nabla v\|^2 \right) \right] \,\mathrm{d}x\mathrm{d}y, \end{aligned} \tag{6.2}$$

where $J_1 \triangleq f_e(I_1; \theta)$ and $J_2 \triangleq f_e(I_2; \theta)$. Hence, this is basically an HS energy function applied to feature images extracted by a neural network. An important thing here is that the feature extraction network itself will be trained during the procedure of the proposed method. In this paper, U-net [54] is used as f_e, with additional padding operations to maintain the size of an image.

U-net is based on a pyramid-like structure as described in Figure 6.5, which can be helpful for finding optical flow as long discussed in the literature. In the proposed framework, (u, v), the optical flow vector, are the inner variables, and I_1 and I_2, the images, are the coefficients. Note that these terms are shown in the above equation as if they are 2-dimensional functions, but in reality, they are handled as 2-dimensional arrays with the same size in the proposed method. Accordingly, $J_2(x+u,\ y+v)$ in the above equation is regarded as a warping operation, where the bilinear interpolation is used. Likewise, the image gradients in the above equation are approximated based on finite differences.

For this problem, the training set K contains M bundles of $(I_1, I_2, u_{gt}, v_{gt})$, where u_{gt} and v_{gt} are ground truth optical flow for I_1 and I_2. The average end point error (EPE) is used as the outer loss, i.e.,

$$L_{\mathrm{op}}(u, v, u', v') = \frac{1}{V} \iint \sqrt{(u-u')^2 + (v-v')^2} \,\mathrm{d}x\mathrm{d}y \tag{6.3}$$

where V is the area of the images (of course, $L_{\mathrm{op}}(u, v, u', v')$ is also computed based on discrete approximation). Algorithm 3 summarizes the whole procedure of the proposed method for optical flow.

Since optical flow is a complicated problem, having good initial points u_0 and v_0 can be vital for the success of the proposed method. If we do not have good initial

Algorithm 3: Learning Objective Function for Optical Flow

```
1  **Input:** Training set $S$, mini-batch size $b$, step size $\gamma$, regularization constant $\lambda$
2  **Initialize:** Network parameters $\theta$
3  for number of training epochs do
4      Initialize $\bar{B} = \{B|B$ is a partition of $S$ with size $b\}$.
5      for $B \in \bar{B}$ do
6          Initialize $Y$ to an empty set.
7          for $(I_1, I_2, u_{gt}, v_{gt}) \in B$ do
8              Initialize $u_0$ and $v_0$.
9              while convergence do
10                 $J_1 = f_e(I_1, \theta)$.
11                 $J_2 = f_e(I_2, \theta)$.
12                 $u_{n+1} = u_n - \gamma \frac{\partial E(u_n, v_n, J_1, J_2)}{\partial u_n}$.
13                 $v_{n+1} = v_n - \gamma \frac{\partial E(u_n, v_n, J_1, J_2)}{\partial v_n}$.
14             end
15             Add $(\hat{u}, \hat{v}, u_{gt}, v_{gt})$ to $Y$.
16             Calculate $\nabla_\theta L_{\text{op}}(\hat{u}, \hat{v}, u_{gt}, v_{gt})$ by backward FPI layer.
17         end
18         Update $\theta$ by $\frac{1}{b} \sum_{(\hat{u},\hat{v},u_{gt},v_{gt}) \in Y} \nabla_\theta L_{\text{op}}(\hat{u}, \hat{v}, u_{gt}, v_{gt})$.
19     end
20 end
```

points, then the possible ranges of u and v that the proposed method has to learn the objective function for can be too large. To avoid this issue, we use existing algorithm, PWC-net [60], as an initializer. In this regard, the product of the proposed method can be viewed as a post-processing scheme to improve the performance further.

U-Net [54] with depth 3 was used as the feature extraction network, which gives the effect of using a 3-level image pyramid. The step size was set to $\gamma = 1.0$, the regularization constant to $\lambda = 180$, and the mini-batch size to $h = 10$.

We used the MPI-Sintel *clean* dataset [19] which is a 3D animation dataset for optical flow. Figure 6.7 shows example images for MPI-Sintel dataset. Its training dataset consists of 23 scenes, $1,064$ image frames, and 1041 ground truth flows, which has been additionally separated to a training set (908) and a validation set (133) in an existing work [24]. In our experiment, this validation set was used as the test set because the original test set does not have any ground truth.

The average EPE of PWC-net for the test set is 2.534 which is the initial error of (u_0, v_0) in our algorithm. We set the number of steps N to 10 which was the maximum possible number we could afford in the GPU memory. The 10 iteration steps improved the average EPE to 2.502 as shown in Figure 6.6. The average EPE decreases smoothly as the iteration proceeds, which suggests that this improvement is not accidental and the proposed method learns a meaningful objective function. If future advances in deep learning hardware allow more memory space, than there is a chance that better performance can be achieved with more gradient descent steps.

Learning Hyperparameters:

When designing the objective function, other traditional methods spend a lot of time tuning hyper-parameters. On the other hand, our algorithms are less sensitive since the network automatically learns a proper objective function. Moreover, we can also attempt to learn the hyper-parameters in the proposed framework, considering them as network parameters. For example, the regularization constant λ in (6.2) can be included in θ as a parameter. Although omitted in this section, we have actually tried training λ several times in this way and have fixed λ to the converged value for every other training attempts. Hyperparameters can be easily chosen in this way or it is also possible to learn them together with other network parameters in each iteration.

6.3 Image generation

Here we show the inverse operation can be easily performed using FPI_GD layer by improving the variational autoencoder. Figure 6.8 shows the structure of variational autoencoder (VAE). In VAE, encoder and decoder are inverse operations with each other. The encoder converts the input data into a smaller dimensional latent variable **z**, and the decoder generates high-dimensional data similar to the input data from **z**. For this, the encoder is composed using conv layers and the decoder is composed using deconv layers. However, conv and deconv layers are not strictly inverse transforms, and ReLU or batch normalization in the middle are not considered. We directly compute the inverse operation of f for given y as follows:

$$\hat{x} = \arg\min_{x} \; \|y - f(x)\|^2. \tag{6.4}$$

To find $\hat{x}$ that satisfies the above equation, we optimize through FPI_GD as follows:

$$\begin{aligned} \bar{f}(x) &= \|y - f(x)\|^2, \\ x_{n+1} &= x_n - \gamma \nabla \bar{f}(x_n). \end{aligned} \tag{6.5}$$

Since the purpose of VAE is to train a decoder that generates images well, we design the FPI to be in the form of the inverse operation of the decoder in order to replace the existing encoder. That is, we replace encoder $E(x)$ to $F(D(x))$ for decoder $D(x)$.

However, since the VAE is also a large network, if you start training at zero base, it takes too much time for training with the current GPU and the number of steps to converge the FPI must be large. So, we conduct an experiment to check feasibility. We limited the maximum number of steps for convergence to 30 for both forward FPI and backward FPI, but instead trained first with the original VAE for a certain epoch to give a guideline. In addition, the latent variable extracted from the original VAE was given for the start input variable x_0 of FPI_GD.

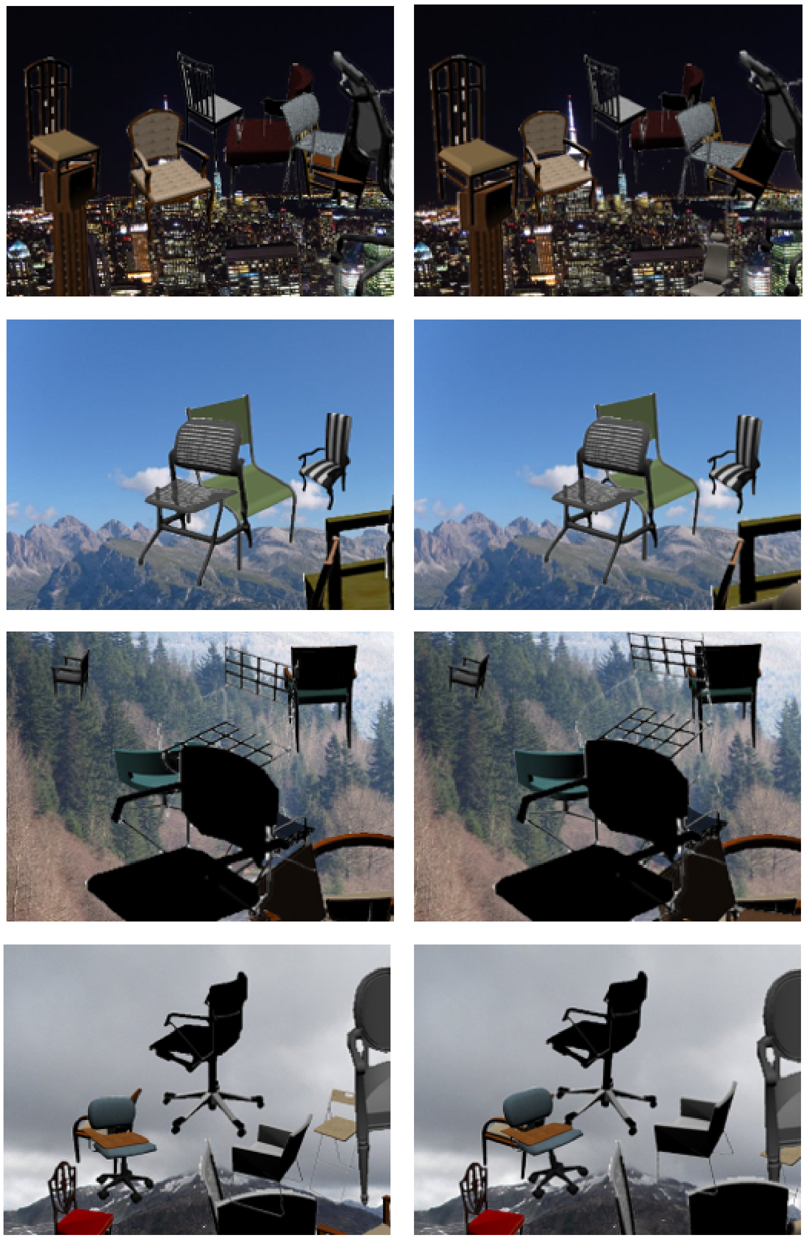


Figure 6.3: Examples for Flying Chair dataset.

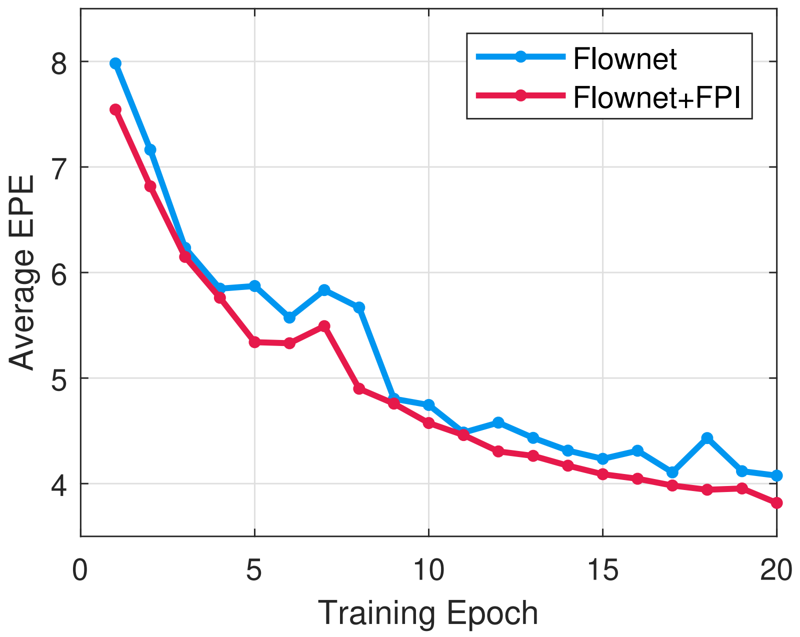


Figure 6.4: Average EPE per epoch (lower is better). A very small FPI layer helps to refine the FlowNet algorithm.

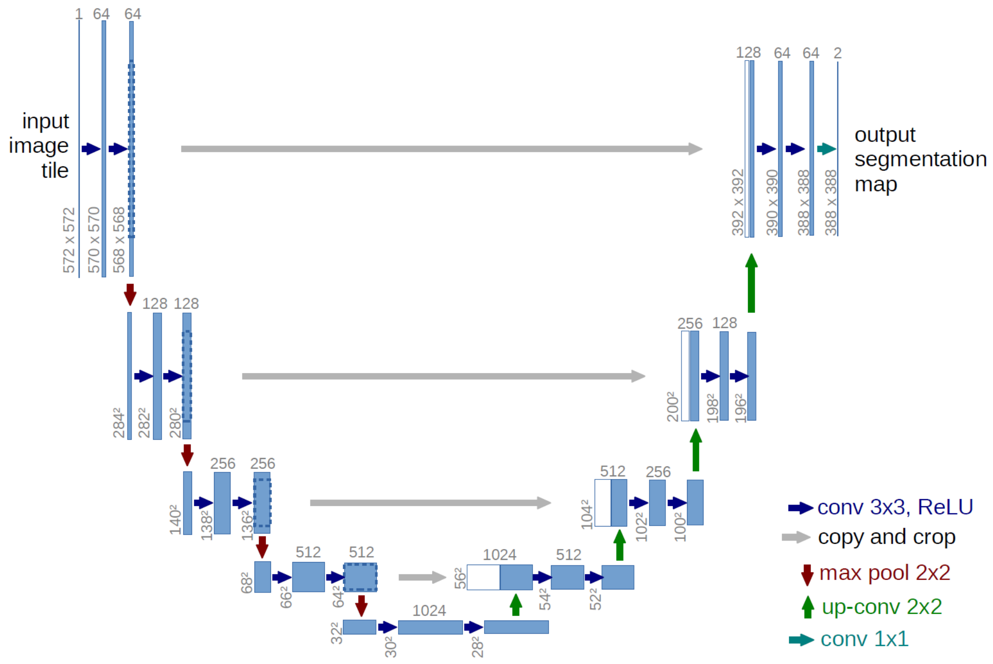


Figure 6.5: Structure of U-Net.

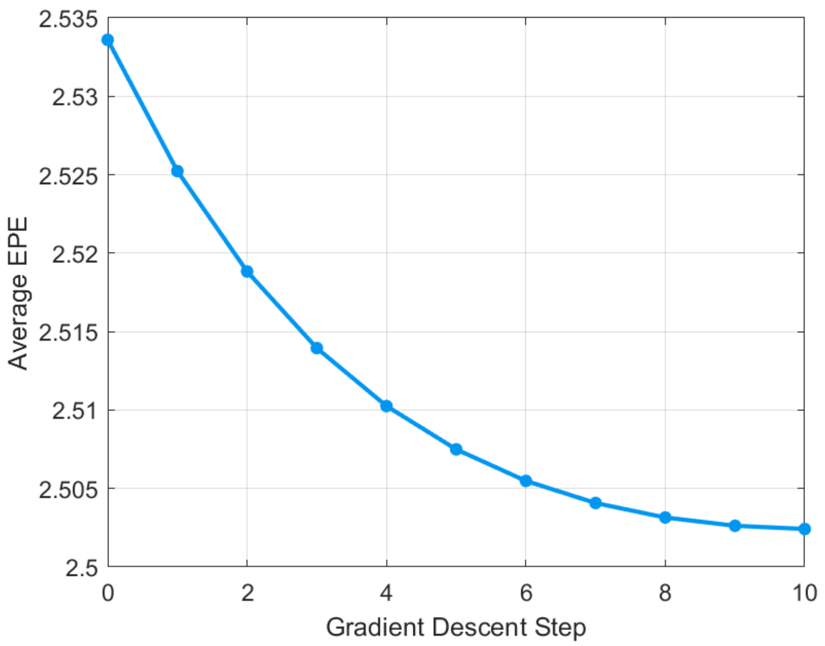


Figure 6.6: Result for Optical Flow. Average EPE for test set decreases through the gradient steps.

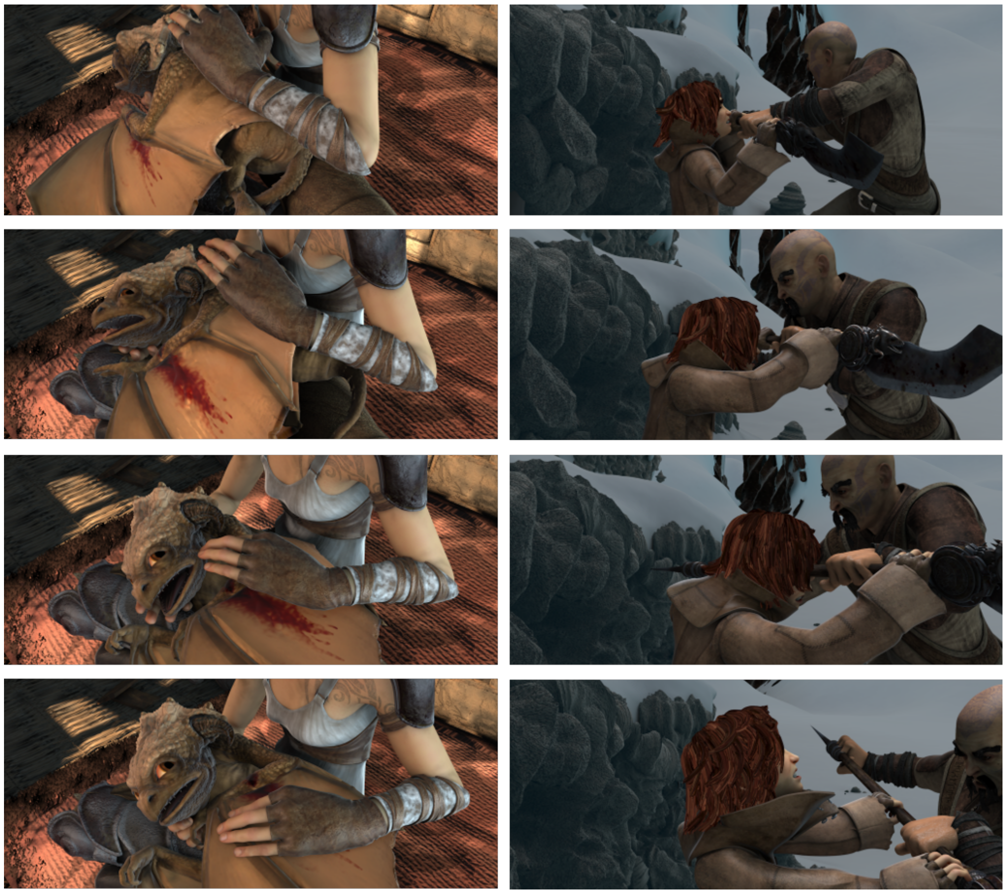


Figure 6.7: Examples for MPI-Sintel dataset.

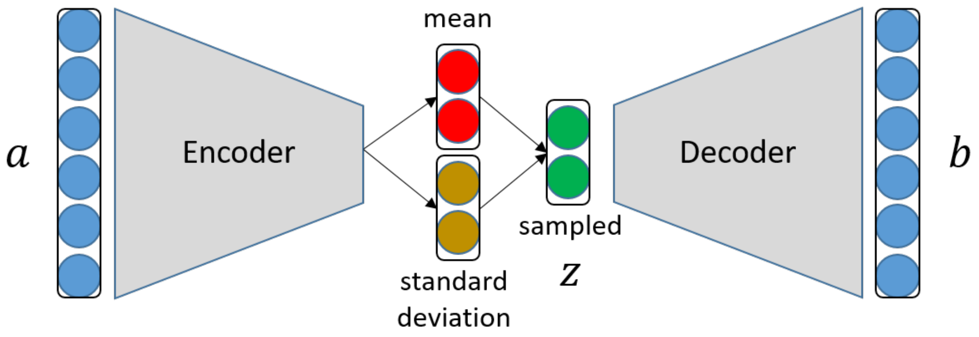


Figure 6.8: Structure of the variational autoencoder. The encoder acts as inverse operation of the decoder.

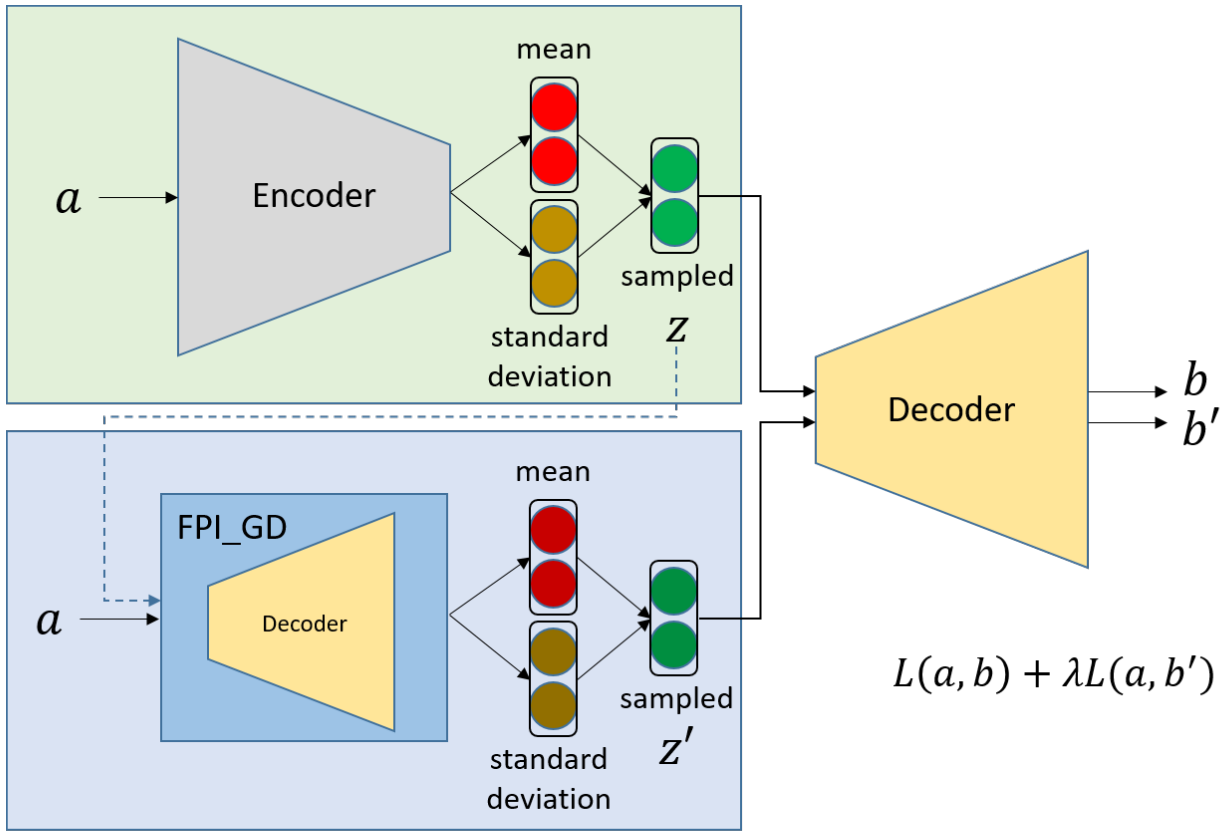


Figure 6.9: Structure of our method with variational autoencoder.

Figure 6.9 shows the entire process of our method combined with VAE. The decoder is trained in two ways and the results of each are compared.

- 60 epochs with original VAE
- 30 epochs with original VAE + 30 epochs with VAE combined with our method

The results for the initial 30 epochs are the same, so we report the results for the later 30 epochs. As a measure, we use the Fréchet inception distance (FID) [30] score, which is widely used to evaluate image generation performance. We use CelebA dataset [45] for evaluation. Figure 6.11 shows the FID score of the original VAE and our method. Despite using a maximum of 30 steps for FPI_GD, our method outperforms the original VAE for every epoch.

Figures 6.14, 6.15 describe image reconstruction and generation results of our method. Our method needs much more training time than original VAE so comparison by epochs may be meaningless. However, since the decoder uses the same structure, the test time is the same. The best performance of our method is 155.324 and the original VAE is 157.717, so it can be seen that our method represents the inverse operation better.

The conv layer and the deconv layer are known to perform the role of inverse transformation between each other well, but our method can be more effective when inverse transformation for various and complex networks or functions is required.

6.4 Implementation details

For the experiments on image denoising and optical flow, only the FPI_NN layer was tested. For the energy function network of the FPI_GD layer to have a scalar output, we attached a mean squared layer at the end of the network. Note that, for all the above layers, the fixed-point iteration variable x was concatenated with the layer input z, and is passed onto g, e.g., for vector inputs

$$g(x, z; \theta) = g([x^T \ z^T]^T; \theta). \tag{6.6}$$

All the training was performed using the Adam optimizer with learning rate 10^{-3}, and no weight decay was used.

6.4.1 Image denoising

The first 500 images of the flying chair dataset [24] were cropped to 180×180 around the center, of which the former 400 images were used to train the networks and the latter 100 were used as test samples. All images were converted to gray scale. 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 the input and the output were gray-scale 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 max norm 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 ten times smaller than the default initialization of PyTorch [25].

6.4.2 Optical flow

We used the FlowNetS [24] structure for this experiment. As in [24], we tested the performance on the flying chair dataset with the same training/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 the 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 both set to two for 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 the 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}. The initial values of the network weights was set 100 times smaller than the default initialization of PyTorch. All other training settings were identical to 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 at the end of FlowNetS, and thus this result is omitted from the experimental results.

6.4.3 Image generation

In this experiment, we used the PyTorch implementation from GitHub [58]. The energy function network structure of FPI_GD was exactly same as the decoder. Unnormalized convergence criterion 5.6 was used. The convergence threshold was 10^{-4} and step size of gradient descent was 0.1. In experiments, the standard deviation was shared between the two methods in order to focus on the overall performance. Hidden dimensions of encoder were 32, 64, 128, 256, 512 and the latent dimension was 128. For evaluation we randomly picked 10,000 images from CelebA dataset and computed the FID score with generated 10,000 images. Size of each images were 64 by 64. The initial values of the energy function network weights of FPI_GD were set 1,000 times smaller than the default initialization of PyTorch.

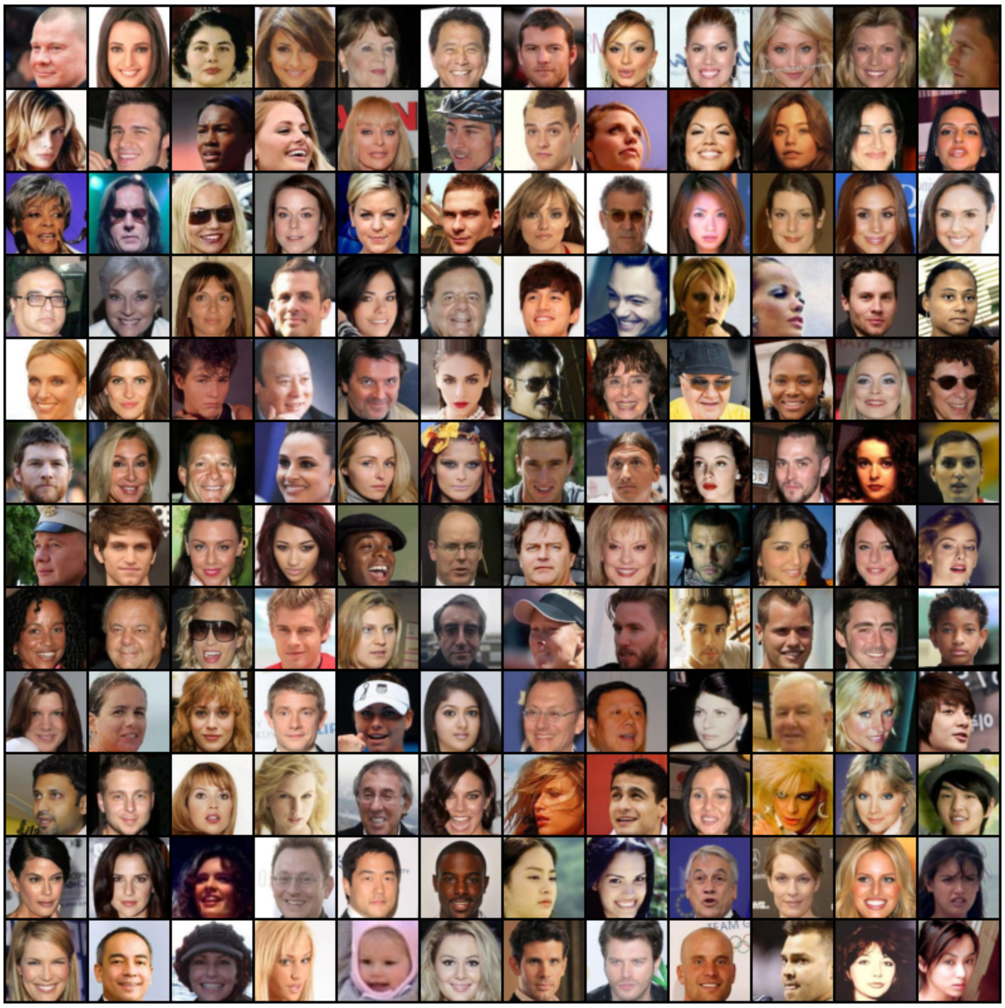


Figure 6.10: Examples of CelebA dataset.

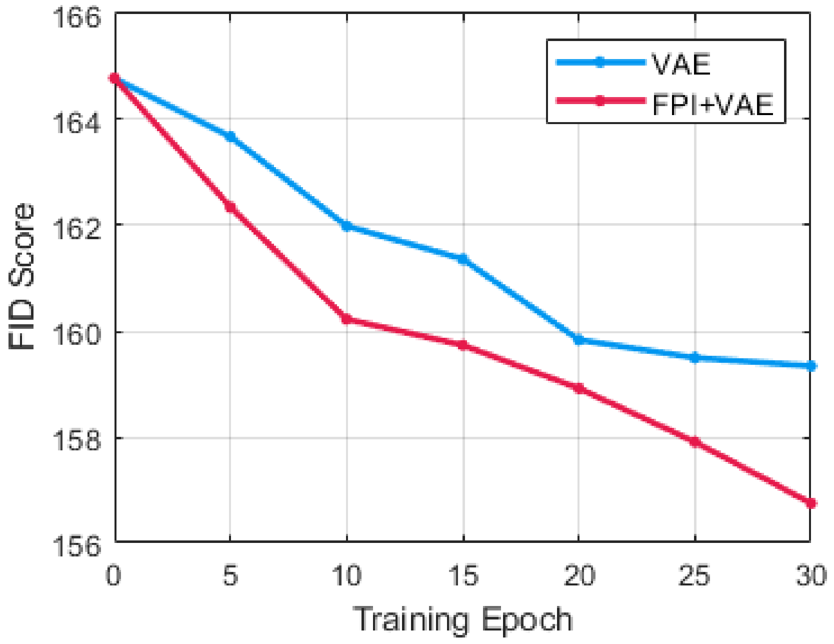


Figure 6.11: FID score for variational autoencoder and our method (lower is better).

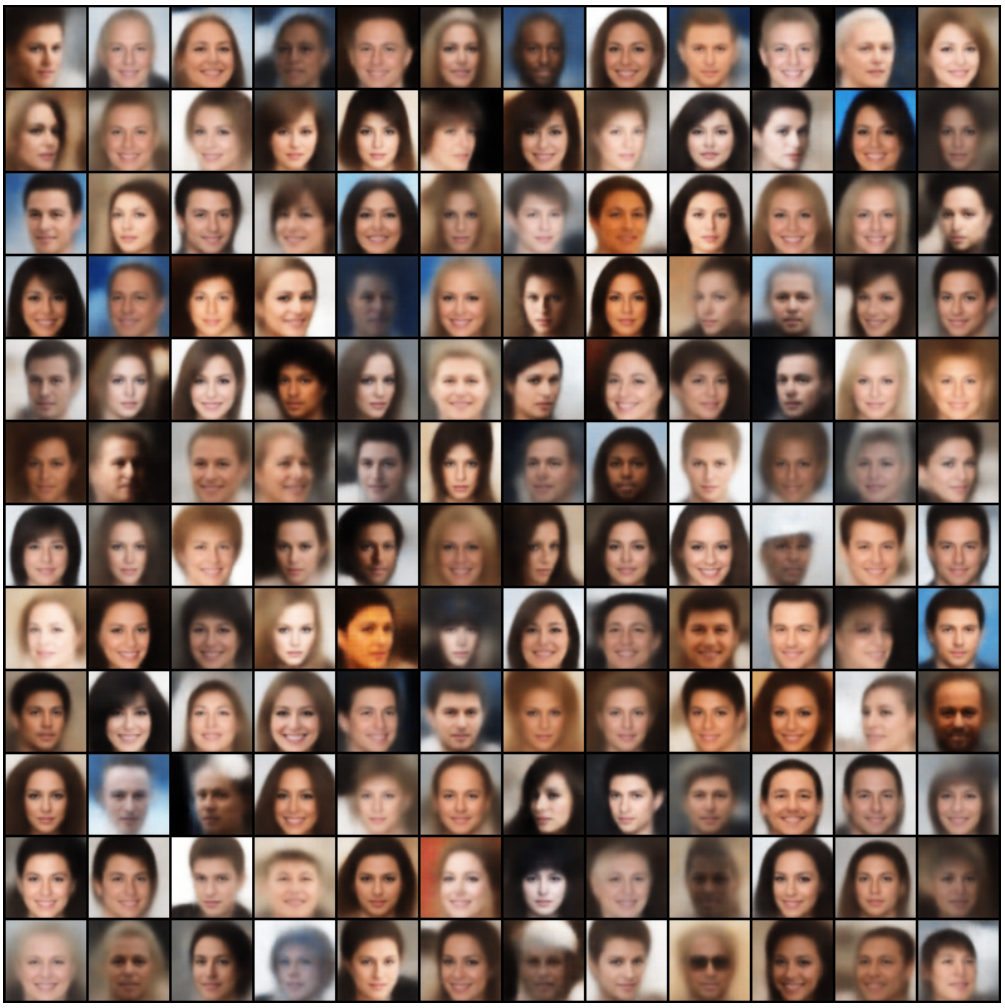


Figure 6.12: Image reconstruction results of original VAE.

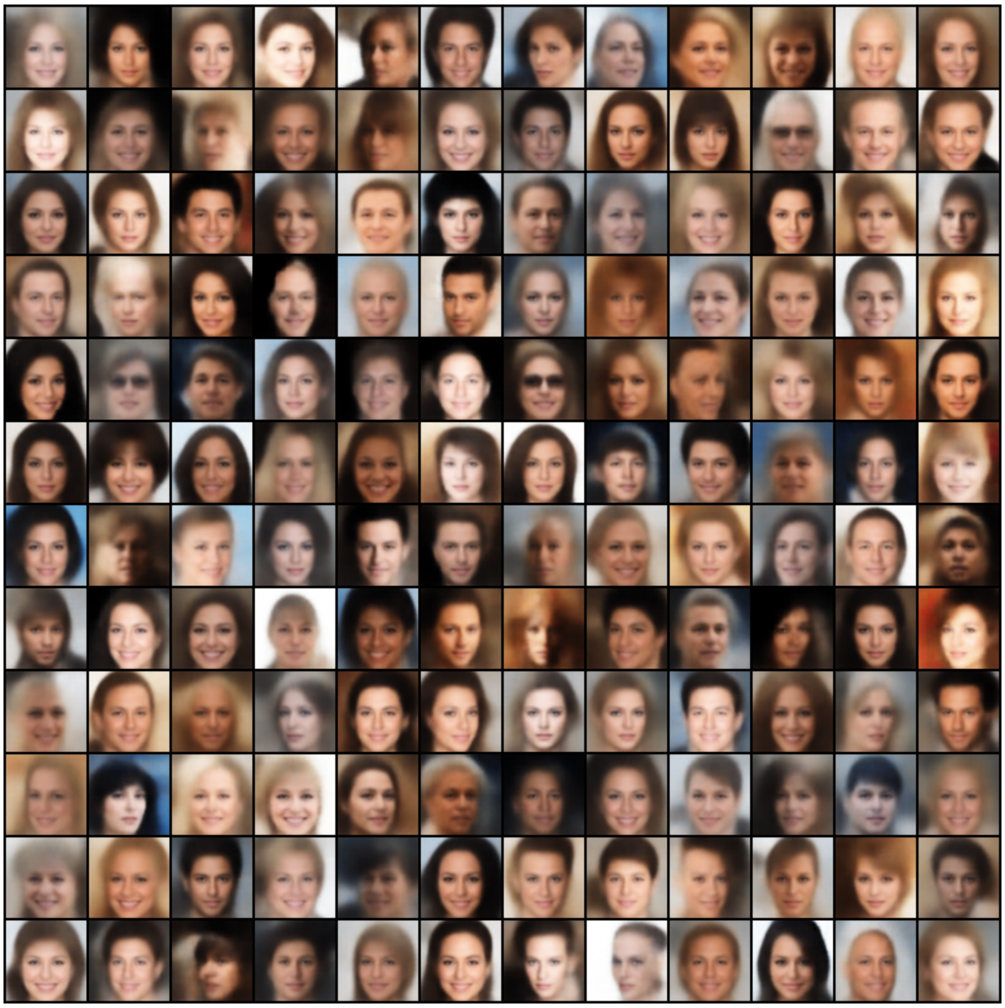


Figure 6.13: Image generation results of original VAE.

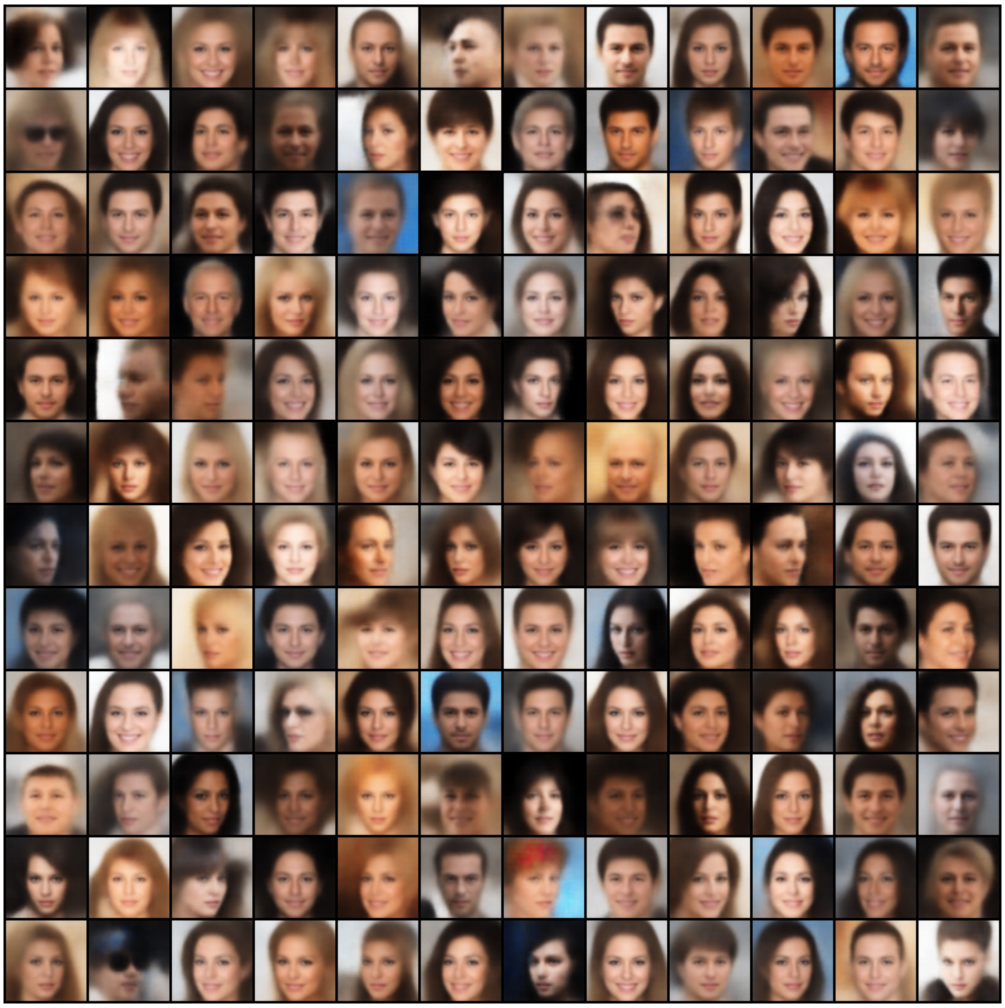


Figure 6.14: Image reconstruction results of our method.

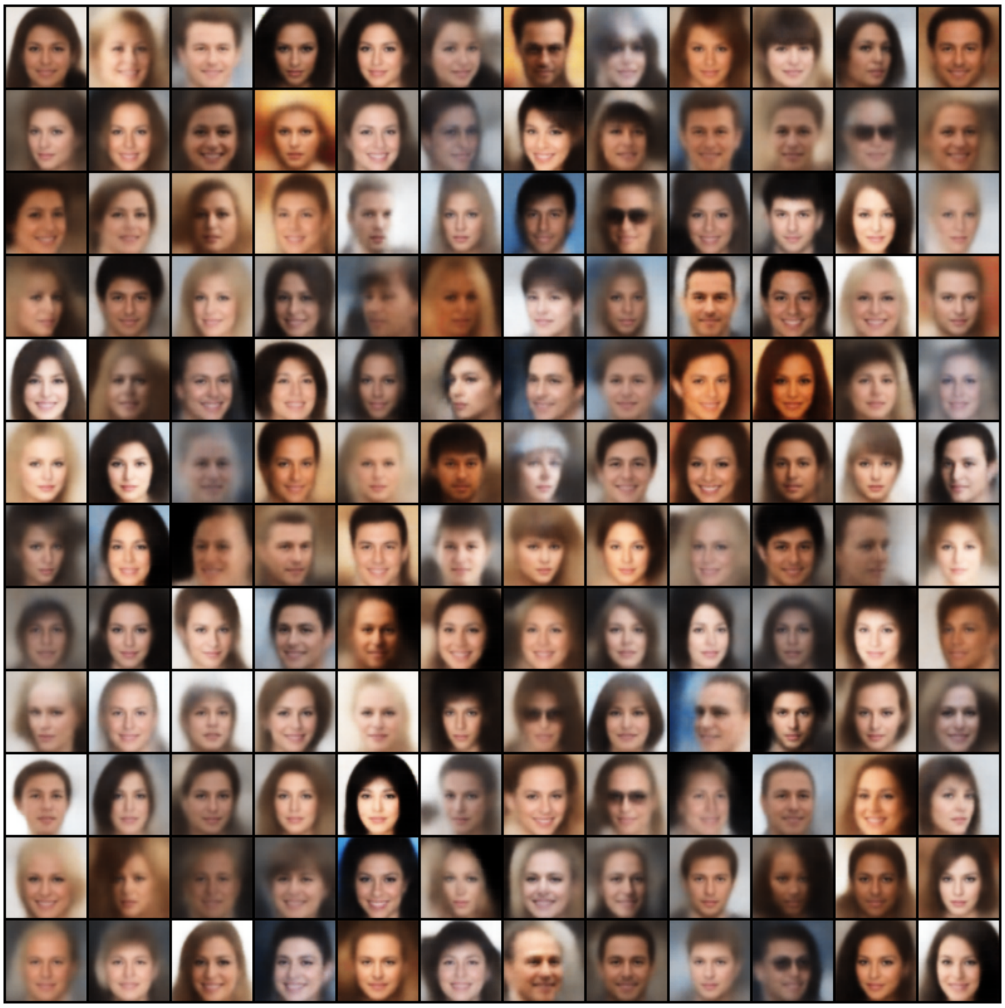


Figure 6.15: Image generation results of our method.

Chapter 7

Conclusion

7.1 Concluding remarks

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

7.2 Broader impact

This work does not present any foreseeable societal consequence for now because it proposes theoretical ideas that can be generally applied to various deep learning structures. However, our method can provide a new direction for various fields of machine learning or computer vision.

Recently, a huge portion of new techniques is developed based on deep learning in various research fields. In many cases, this leads to rewriting a large part of the traditional methodologies, because deep networks bear quite different structures from traditional algorithms. During the process, much of the conventional wisdom found in existing theories are being re-discovered based on raw datasets. This induces a high economic cost in developing new technologies. Our method can provide an alternative to this trend by incorporating the existing mechanisms in many iterative algorithms, which can reduce the development costs. There already have been many studies that combine deep learning with more complicated models such as SMPL, however, one usually has to derive the backpropagation formula separately for each method, which introduces considerable difficulties and, as a result, development costs. However, our method can be applied universally to many types of iterative algorithms, so the consilience between various models from different fields can be stimulated.

Another possible consequence of the proposed method is that it might also expand the application of deep learning in non-GPU environments. As demonstrated in the experiments, the introduction of an FPI operation in a deep network can achieve similar performance with a much simpler structure, at the expense of an iterative calculation. This can be helpful for using deep networks in many resource-limited environments and may accelerate the trend of ubiquitous deep learning.

7.3 Limitations and future work

As the structure of g becomes more complex, it is difficult to make g contraction mapping and the probability of divergence increases. However, if we make it a layer-by-layer contraction mapping, the performance decreases. In other words, it seems necessary to consider the structure of the components of g in order to obtain good performance while using a complex structure in future studies. In addition, there were many experiments that could not be performed because of the long training time, and the number of iterations had to be limited. However, if the hardware develops, various results will be obtained.

Since this research area has not been studied much yet, there is a lot of potential for improvement. First of all, new structures of g and their applications are worth studying. For a simple and intuitive example, we can use another gradient-based algorithm like Adam [18] instead of gradient descent in FPI_GD. Using multiple input sources can be an interesting future research direction. We use single input source x in this work but multiple input sources and alternate optimizations using multiple g can yield new effects. Learning the initial value x_0, which is initialized to a zero matrix (or zero vector) for now, based on z for improving the efficiency can also be an interesting research direction.

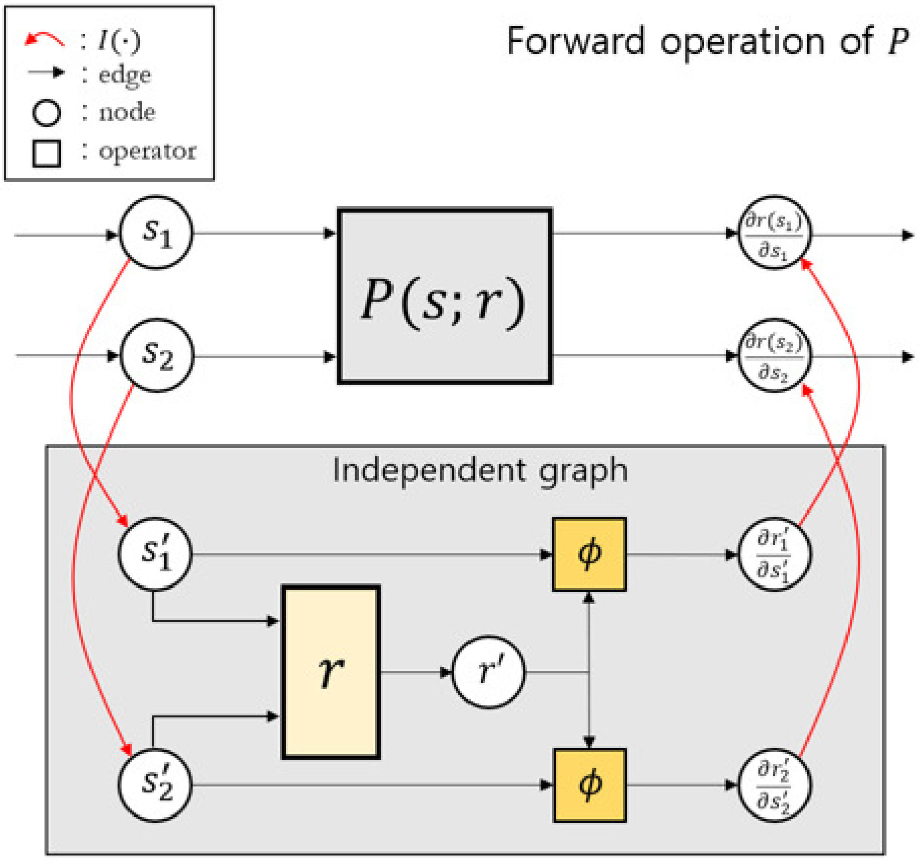
The purpose of this paper is to propose a new structure and to show that it can be learned through the universal backpropagation formula. However, if we want to apply this study in the future to obtain state-of-the-art performance in a specific field, it would be better to use empirical information such as KKT conditions.

Chapter 8

Appendix

8.1 Figures of partial differentiation and the backward FPI layer

The following figures (Figure 8.1, 8.2, 8.3) 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 a usual autograd framework without any explicit Jacobian computation.



: $I(\cdot)$
: edge
: node
: operator
Forward operation of P
s_1
s_2
$P(s;r)$
$\frac{\partial r(s_1)}{\partial s_1}$
$\frac{\partial r(s_2)}{\partial s_2}$
Independent graph
s'_1
s'_2
r
r'
ϕ
ϕ
$\frac{\partial r'_1}{\partial s'_1}$
$\frac{\partial r'_2}{\partial s'_2}$

Figure 8.1: Forward operation of partial differentiation.

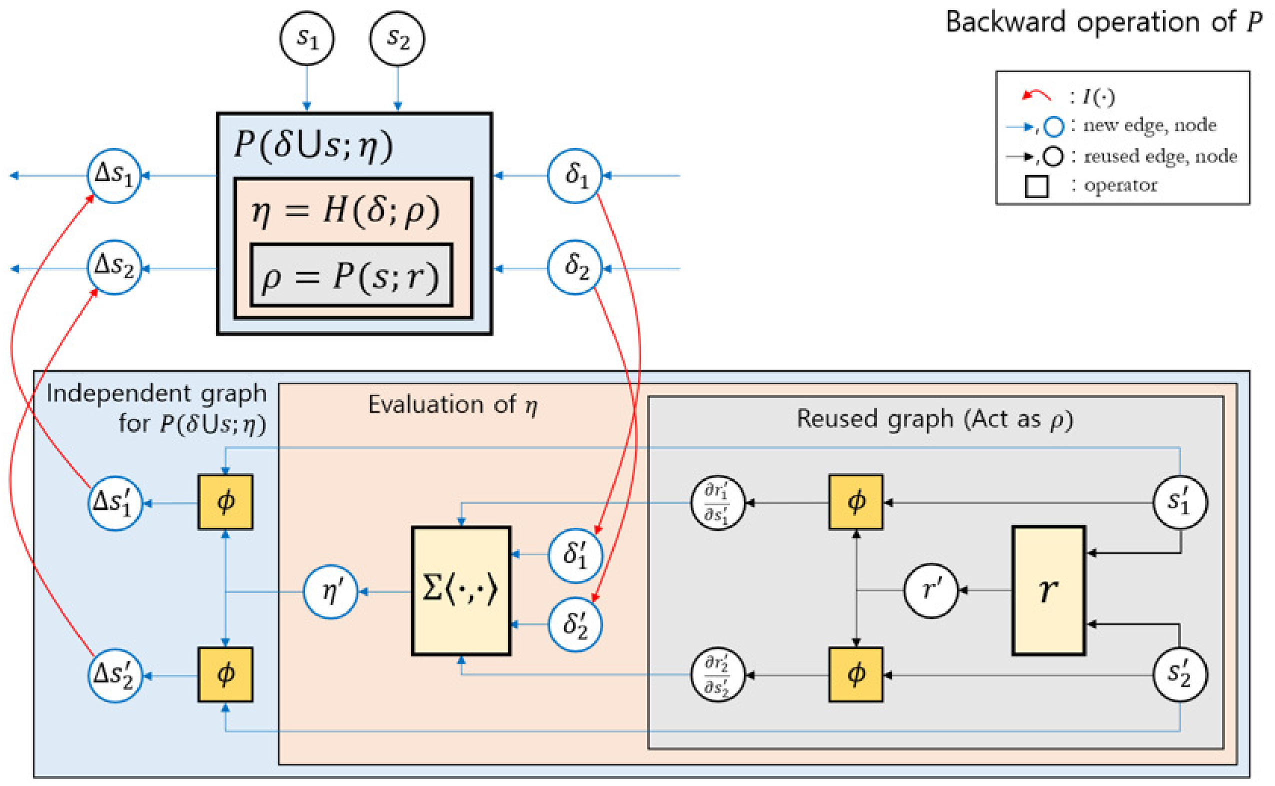


Figure 8.2: Backward operation of partial differentiation.

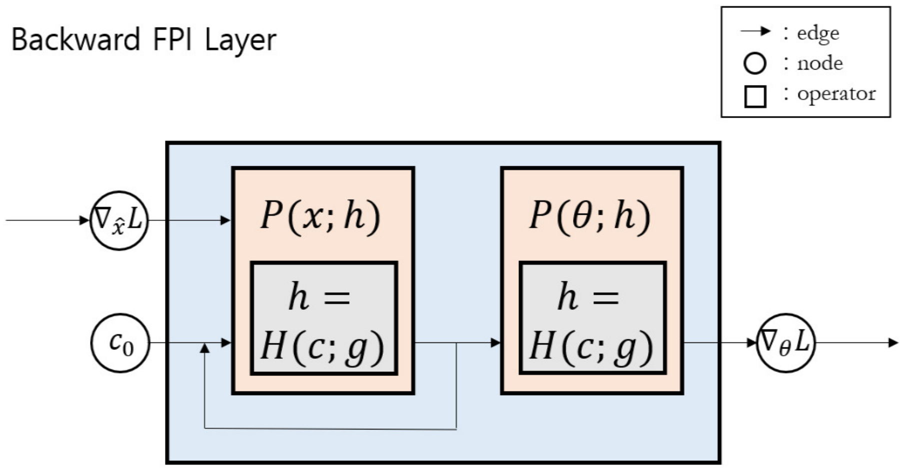


Figure 8.3: The backward FPI layer.

8.2 Additional lemma 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 2. *If the matrix norm $\|A\| < 1$, then the linear transform by weight A, i.e., $f(x) = Ax + b$, is a contraction mapping.*

Proof.

$$\begin{aligned}\frac{\|f(x_1)-f(x_2)\|}{\|x_1-x_2\|} &= \frac{\|Ax_1-Ax_2\|}{\|x_1-x_2\|} \\ &= \frac{\|A(x_1-x_2)\|}{\|x_1-x_2\|} \\ &\leq \frac{\|A\|\cdot\|x_1-x_2\|}{\|x_1-x_2\|} \\ &= \|A\| \\ &\leq k < 1\end{aligned} \tag{8.1}$$

By the definition, f is a contraction mapping.

□

In section 3.4,

$$\|J_g(\hat{x})\| \leq k < 1, \tag{8.2}$$

which means that the linear transform with weight matrix $J_g(\hat{x})$ is a contraction mapping by *Lemma 2*. Therefore, (10) of the backward FPI in the main paper is a contraction mapping.

Bibliography

[1] Abbeel, Pieter and Ng, Andrew Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

[2] Agrawal, Akshay, Amos, Brandon, Barratt, Shane, Boyd, Stephen, Diamond, Steven, and Kolter, J Zico. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, pp. 9558–9570, 2019.

[3] Agrawal, Akshay, Barratt, Shane, Boyd, Stephen, Busseti, Enzo, and Moursi, Walaa M. Differentiating through a conic program. *arXiv preprint arXiv:1904.09043*, 2019.

[4] Almeida, Luis B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *IEEE International Conference on Neural Networks, 1987*, pp. 609–618, 1987.

[5] Amos, Brandon and Kolter, J Zico. 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.

[6] Amos, Brandon, Xu, Lei, and Kolter, J Zico. Input convex neural networks. In

Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 146–155. JMLR. org, 2017.

[7] Andrychowicz, Marcin, Denil, Misha, Gomez, Sergio, Hoffman, Matthew W, Pfau, David, Schaul, Tom, Shillingford, Brendan, and De Freitas, Nando. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

[8] Aneja, Jyoti, Schwing, Alexander, Kautz, Jan, and Vahdat, Arash. Ncp-vae: Variational autoencoders with noise contrastive priors. *arXiv preprint arXiv:2010.02917*, 2020.

[9] Bai, Shaojie, Kolter, J Zico, and Koltun, Vladlen. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pp. 688–699, 2019.

[10] Banach, Stefan. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math*, 3(1):133–181, 1922.

[11] Beardsell, Philippe and Hsu, Chih-Chao. *Structured Prediction with Deep Value Networks*, 2020. URL `https://github.com/philqc/deep-value-networks-pytorch`.

[12] Belanger, David and McCallum, Andrew. Structured prediction energy networks. In *International Conference on Machine Learning*, pp. 983–992, 2016.

[13] Belanger, David, Yang, Bishan, and McCallum, Andrew. 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.

[14] Black, Michael J and Anandan, Paul. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer vision and image understanding*, 63(1):75–104, 1996.

[15] Böhm, Vanessa and Seljak, Uroš. Probabilistic auto-encoder. *arXiv preprint arXiv:2006.05479*, 2020.

[16] Brock, Andrew, Donahue, Jeff, and Simonyan, Karen. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[17] Brox, Thomas, Bruhn, Andrés, Papenberg, Nils, and Weickert, Joachim. High accuracy optical flow estimation based on a theory for warping. In *European conference on computer vision*, pp. 25–36. Springer, 2004.

[18] Burden, Richard and Faires, JD. *Numerical analysis*. Cengage Learning, 2004.

[19] Butler, Daniel J, Wulff, Jonas, Stanley, Garrett B, and Black, Michael J. A naturalistic open source movie for optical flow evaluation. In *European conference on computer vision*, pp. 611–625. Springer, 2012.

[20] Chen, Xi, Kingma, Diederik P, Salimans, Tim, Duan, Yan, Dhariwal, Prafulla, Schulman, John, Sutskever, Ilya, and Abbeel, Pieter. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

[21] Cybenko, George. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[22] Djolonga, Josip and Krause, Andreas. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems*, pp. 1013–1023, 2017.

[23] Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[24] Fischer, Philipp, Dosovitskiy, Alexey, Ilg, Eddy, Häusser, Philip, Hazırbaş, Caner, Golkov, Vladimir, Van der Smagt, Patrick, Cremers, Daniel, and Brox,

Thomas. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[25] Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[26] Gong, Xinyu, Chang, Shiyu, Jiang, Yifan, and Wang, Zhangyang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3224–3234, 2019.

[27] Goodfellow, Ian J, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[28] Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, and Courville, Aaron. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[29] Gygli, Michael, Norouzi, Mohammad, and Angelova, Anelia. 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.

[30] Heusel, Martin, Ramsauer, Hubert, Unterthiner, Thomas, Nessler, Bernhard, and Hochreiter, Sepp. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017.

[31] Hinton, Geoffrey, Srivastava, Nitish, and Swersky, Kevin. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14, 2012.

[32] Hinton, Geoffrey E and Salakhutdinov, Ruslan R. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[33] Horn, Berthold KP and Schunck, Brian G. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[34] Hornik, Kurt. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[35] Hui, Tak-Wai, Tang, Xiaoou, and Change Loy, Chen. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8981–8989, 2018.

[36] Ilg, Eddy, Mayer, Nikolaus, Saikia, Tonmoy, Keuper, Margret, Dosovitskiy, Alexey, and Brox, Thomas. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2462–2470, 2017.

[37] Jiang, Borui, Luo, Ruixuan, Mao, Jiayuan, Xiao, Tete, and Jiang, Yuning. Acquisition of localization confidence for accurate object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–799, 2018.

[38] Katakis, Ioannis, Tsoumakas, Grigorios, and Vlahavas, Ioannis. Multilabel text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD*, volume 18, pp. 5, 2008.

[39] Khamsi, Mohamed A and Kirk, William A. *An introduction to metric spaces and fixed point theory*, volume 53. John Wiley & Sons, 2011.

[40] Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[42] LeCun, Yann, Chopra, Sumit, Hadsell, Raia, Ranzato, M, and Huang, F. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[43] Li, Ke and Malik, Jitendra. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[44] Liao, Renjie, Xiong, Yuwen, Fetaya, Ethan, Zhang, Lisa, Yoon, KiJung, Pitkow, Xaq, Urtasun, Raquel, and Zemel, Richard. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pp. 3082–3091, 2018.

[45] Liu, Ziwei, Luo, Ping, Wang, Xiaogang, and Tang, Xiaoou. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[46] Makhzani, Alireza, Shlens, Jonathon, Jaitly, Navdeep, Goodfellow, Ian, and Frey, Brendan. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[47] Ng, Andrew et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[48] Ng, Andrew Y, Russell, Stuart J, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pp. 2, 2000.

[49] Papenberg, Nils, Bruhn, Andrés, Brox, Thomas, Didas, Stephan, and Weickert, Joachim. Highly accurate optic flow computation with theoretically justified warping. *International Journal of Computer Vision*, 67(2):141–158, 2006.

[50] Peng, Yue, Deng, Bailin, Zhang, Juyong, Geng, Fanyu, Qin, Wenjie, and Liu, Ligang. Anderson acceleration for geometry optimization and physics simulation. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[51] Pineda, Fernando J. Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229, 1987.

[52] Ramachandran, Deepak and Amir, Eyal. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pp. 2586–2591, 2007.

[53] Razavi, Ali, Oord, Aäron van den, Poole, Ben, and Vinyals, Oriol. Preventing posterior collapse with delta-vaes. *arXiv preprint arXiv:1901.03416*, 2019.

[54] Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

[55] Roth, Stefan, Lempitsky, Victor, and Rother, Carsten. Discrete-continuous optimization for optical flow estimation. In *Statistical and Geometrical Approaches to Visual Motion Analysis*, pp. 1–22. Springer, 2009.

[56] Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016.

[57] Sohn, Kihyuk, Lee, Honglak, and Yan, Xinchen. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28:3483–3491, 2015.

[58] Subramanian, A.K. Pytorch-vae. `https://github.com/AntixK/PyTorch-VAE`, 2020.

[59] Sun, Deqing, Roth, Stefan, and Black, Michael J. Secrets of optical flow estimation and their principles. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pp. 2432–2439. IEEE, 2010.

[60] Sun, Deqing, Yang, Xiaodong, Liu, Ming-Yu, and Kautz, Jan. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8934–8943, 2018.

[61] Tsochantaridis, Ioannis, Hofmann, Thomas, Joachims, Thorsten, and Altun, Yasemin. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 104, 2004.

[62] Vahdat, Arash and Kautz, Jan. Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*, 2020.

[63] Wang, Po-Wei, Donti, Priya L, Wilder, Bryan, and Kolter, Zico. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv preprint arXiv:1905.12149*, 2019.

[64] Wang, Weiran and Carreira-Perpinán, Miguel A. Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application. *arXiv preprint arXiv:1309.1541*, 2013.

[65] Zach, Christopher, Pock, Thomas, and Bischof, Horst. A duality based approach for realtime tv-l1 optical flow. In *Joint pattern recognition symposium*, pp. 214–223. Springer, 2007.

[66] Zeiler, Matthew D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[67] Ziebart, Brian D, Maas, Andrew L, Bagnell, J Andrew, and Dey, Anind K. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pp. 1433–1438. Chicago, IL, USA, 2008.

초록

최근 여러 연구에서 기존 신경망 레이어들이 이용할 수 없던 제약 조건들을 인코딩하기 위해, 심층 신경망을 설계할 때 최적화 문제를 활용하는 방법을 제안하였다. 그러나 이러한 방법들은 아직 초기 단계이며 역전파 공식을 도출하기 위해 KKT 조건 분석과 같은 특별한 처리가 필요하다. 본 논문에서는 딥 네트워크에서 더 복잡한 연산을 쉽게 사용할 수 있도록 고정점 반복 레이어라고 하는 새로운 형태의 레이어을 제안한다. 반복적 역전파 방법에 기반하여, 역방향 고정점 반복 레이어도 제안한다. 그러나 반복적 역전파와 달리, 역방향 고정점 반복 레이어는 야코비 행렬을 직접 계산하지 않고 작은 네트워크 모듈을 이용하여 그래디언트를 얻는다. 또한 본 논문에서는 두 가지 실용적인 방법(FPI_NN과 FPI_GD)을 제안한다. FPI_NN은 작은 신경망 모듈을 반복하여 업데이트하며, 직관적이고 간단하다. FPI_GD는 경사하강법을 반복하여 업데이트하며, 최근들어 연구된 에너지 네트워크들의 효율적인 학습 방법이다. 반복적 역전파 방법 및 관련 연구들이 실제 사례에 적용된 경우가 거의 없었는데, 본 논문의 실험은 고정점 반복 레이어가 이미지 노이즈 제거, 광학 흐름 측정, 다중 라벨 분류 및 이미지 생성과 같은 실제 문제에 성공적으로 적용될 수 있음을 보여준다.

Keywords: 고정점 반복, 경사 하강법, 미분가능 레이어, 반복적 역전파, 에너지 네트워크, 딥러닝 아키텍쳐

Student Number: 2014-21619