Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Yixiao Chen April 8, 2023

Efficient Training of Input Convex Neural Networks Using Variable Projection

by

Yixiao Chen

Lars Ruthotto
Adviser

Mathematics

Lars Ruthotto

Adviser

Robert Roth

Committee Member

Ruoxuan Xiong

Committee Member

2023

Efficient Training of Input Convex Neural Networks Using Variable Projection

By

Yixiao Chen

Lars Ruthotto

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Mathematics

2023

Abstract

Efficient Training of Input Convex Neural Networks Using Variable Projection
By Yixiao Chen

Normalizing flows are deep generative models that construct a diffeomorphic mapping between a simple reference distribution and a complex probability distribution.

Previous studies have utilized the gradient of input convex neural networks (ICNNs) to represent the diffeomorphism, which guarantees invertibility for any network weights.

Moreover, the resulting mapping constitutes a unique optimal transformation that minimizes transportation costs, as dictated by optimal transport theory.

Training ICNNs presents a challenging non-convex optimization problem, except for weights of the last layer.

To address this issue, this thesis proposes a training approach for ICNNs using variable projection (VarPro).

The proposed method takes advantage of the affine mapping in the last layer of ICNNs, which preserves convexity of the network.

Empirical results based on a two-dimensional synthetic dataset demonstrate that VarPro achieves a lower test loss and requires fewer gradient evaluations compared to the mini-batch gradient descent method Adam.

Efficient Training of Input Convex Neural Networks Using Variable Projection

By

Yixiao Chen

Lars Ruthotto

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Mathematics

2023

Acknowledgements

I am honored to express my deep appreciation to Emory University for providing me with a wealth of academic resources and invaluable research opportunities. I am incredibly grateful to my esteemed committee members, Dr. Ruthotto, Dr. Roth, and Dr. Xiong, for their unwavering support and guidance throughout my research project. Dr. Ruthotto's continuous support and willingness to answer all my queries unreservedly were crucial in overcoming the challenges I encountered during the project. His dedication to the field and willingness to share his knowledge and experience with others is truly inspiring, and I have learned so much from our interactions. Dr. Roth's direct research experience during my junior year and continued support throughout this project have been invaluable in shaping my research skills. Dr. Xiong's inspiration to engage in machine learning research and her insightful advice for conducting academic research were significant contributions to my success, and I am grateful for her mentorship.

I am also indebted to Dr. Verma for his guidance and constructive feedback, which played a vital role in the successful completion of my project. His keen insights and analytical skills have helped me refine my research question and methodology. Additionally, I extend my appreciation to my best friend, Oliver Wang, who has been an invaluable study and research partner throughout the project. His support and encouragement were instrumental in keeping me motivated.

I also wish to acknowledge the exceptional faculty, including Dr. Xi, Dr. Just, Dr. Mayo, and Dr. Ananth, for their encouragement of academic research and providing me with opportunities to explore different fields of study. Their mentorship and guidance have been instrumental in shaping my academic journey.

Finally, I want to express my heartfelt gratitude to my family for their unwavering love and support, which has been a constant source of strength and inspiration throughout my academic journey. I am incredibly fortunate to have such supportive and loving family members who have always encouraged me to pursue my passions.

Once again, thank you to everyone who has played a role in my success. Your support and guidance have been invaluable, and I am grateful for the opportunities you have provided me to grow and excel.

Contents

List of Figures

Chapter 1

Introduction

Generative modeling has a wide range of applications in computer vision [3], natural language processing [15], image generation [4], and many more. For instance, image generation, such as creating synthetic celebrity portraits, entails producing new images that closely resemble a limited quantity of available data [4]. Bridging the gap between these diverse applications, deep generative models have emerged as a powerful tool to capture the underlying structure and features of the data.

Deep generative models utilize deep neural networks to learn complex probability distributions from a finite number of independent and identically distributed samples. The primary objective of such models is not only to generate additional samples that are similar to the available data but also to estimate the density of the underlying distribution. Normalizing flows are one type of deep generative models that leverage the concept of a diffeo-

morphic[1] and orientation-preserving mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$ to model the generator [14]. By utilizing this mapping, we can generate new samples from a reference distribution via the generateor, while also estimating the density of the complex probability distribution with the inverse of the generator; see Figure 1.1 for an illustration.

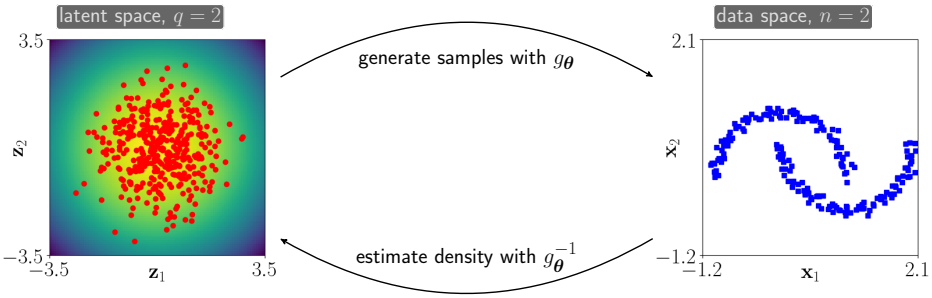

Figure 1.1: Normalizing flows between target and reference distribution[14]. Generate new samples from the target distribution using $g_{\boldsymbol{\theta}}$ and estimate the target density with $g_{\boldsymbol{\theta}}^{-1}$, where $\boldsymbol{\theta}$ are weights of the neural networks.

A considerable amount of existing literature has attempted to represent the diffeomorphic mapping. One notable proposal by [7] involves parameterizing normalizing flows via the gradient of input convex neural networks (ICNNs) using optimal transport theory. Given that ICNNs provides a universal approximation of convex functions, their gradient establishes a diffeomorphism for any choice of network weights, ensuring the invertibility of the map. Additionally, this mapping approximates the Kantorovich potentials, which provide optimality with respect to the lowest transportation cost [7].

ICNNs' weights are typically updated following the maximum likelihood

[1] A diffeomorphic function $g : \mathbb{R}^n \to \mathbb{R}^n$ is a function that is invertible and both g and g^{-1} are continuously differentiable.

training scheme. However, training ICNNs is significantly challenged by large number of weights and the non-convexity of the loss function in all layers except for the last one. This is due to the nonlinear nature of ICNNs, and the preservation of convexity is only guaranteed for weights that enter in an affine manner. To address the challenge of training ICNNs, we draw inspiration from recent work on training deep neural networks for regression and classification [12, 10] and exploit the separable structures of ICNNs.

1.1 Contribution & Outline

This thesis proposes a novel approach for training ICNNs using variable projection (VarPro). The proposed method leverages the affine mapping in the last layer of ICNNs, which guarantees convexity of the objective function in the last layer's weights. To be specific, the last layer's weights are updated initially by solving a smooth convex optimization problem of moderate size. Since some components of the weights of each layer of ICNNs are constrained to be non-negative to preserve convexity, we implement an interior point method that utilizes a log barrier to approximate the constraint in the objective function.

Based on empirical results obtained from a two-dimensional synthetic dataset, the effectiveness and efficiency of the proposed VarPro approach is demonstrated. Specifically, the results show that VarPro outperforms the mini-batch gradient descent method, Adam, in terms of achieving a lower

test loss and requiring fewer gradient evaluations.

This thesis will consist of several key chapters. Chapter 2 will provide an in-depth background on convex potential flow and variable projection, which will serve as the foundation for the proposed training approach. In Chapter 3, the proposed approach for training ICNNs will be elaborately explained, highlighting the efficient implementation techniques used. The results of numerical experiments conducted using the proposed approach will be presented in Chapter 4. In Chapter 5, future directions and potential areas of improvement for the proposed approach will be discussed. Finally, Chapter 6 will provide a comprehensive conclusion, summarizing the contributions of the thesis and their significance in the context of the broader field.

Chapter 2

Background

In this chapter, we delve into the brief introduction of convex potential flows (CP-flow) proposed by [7], which serves as the main framework for our study. We then describe an in-depth explanation of the training problems associated with it. Our primary focus is on training ICNNs, which serve as the parametrization for normalizing flows in CP-flow. We also explore the existing work on utilizing variable projection for training deep neural networks in regression and classification problems. We note that the last layer of ICNNs, like most state-of-the-art deep neural networks, comprises an affine mapping. This leads to our main contribution, which proposes a numerically efficient method for training ICNNs.

2.1 Convex Potential Flows

A normalizing flow is a sequence of transformations that maps a reference distribution η to a complex target distribution $\mathcal{X}$:

$$g_{\boldsymbol{\theta}}(\mathbf{z}) = f_{\boldsymbol{\theta}_k} \circ f_{\boldsymbol{\theta}_{k-1}} \circ \ldots \circ f_{\boldsymbol{\theta}_1}(\mathbf{z}).$$

where $\mathbf{z} \sim \eta$, $f_{\boldsymbol{\theta}_k} : \mathbb{R}^n \to \mathbb{R}^n$ are the transformations, and k is the depth of the network. Each transformation must be invertible and differentiable with a tractable Jacobian determinant to allow for efficient sampling and likelihood computation [14].

To represent each transformation, [7] proposed using input convex neural networks (ICNNs). Specifically, they model the inverse of the generator using gradient of ICNNs $\Phi_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}$ with having weights $\boldsymbol{\theta} \in \mathbb{R}^p$. The weights are then trained such that the $\nabla\Phi_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^n$ ensures that $\nabla\Phi_{\boldsymbol{\theta}}(\mathbf{x}) \sim \eta$, where $\mathbf{x} \sim \mathcal{X}$. The gradient of $\Phi_{\boldsymbol{\theta}}$ is a diffeomorphism for any choice of $\boldsymbol{\theta}$ due to the convexity of the function. Moreover, the mapping approximates the Kantorovich potentials, which ensures optimality according to optimal transport theory. After training, new samples can be produced using $g_{\boldsymbol{\theta}}(\mathbf{z}) := \nabla\Phi_{\boldsymbol{\theta}}^{-1}(\mathbf{z})$, which leads to a convex optimization problem described in detail in [7].

The normalizing flows that [7] used are parameterized by l-layer (fully)

input convex neural networks (ICNNs) introduced in [1], which take the form:

$$\mathbf{u}_i = \sigma_i(\mathbf{L}_i^+ \mathbf{u}_{i-1} + \mathbf{K}_i \mathbf{x} + \mathbf{b}_i), \quad \text{for } i = 1 \dots l-1, \tag{2.1}$$

$$\Phi_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{u}_\ell = \mathbf{L}_l^+ \mathbf{u}_{l-1} + \mathbf{K}_l \mathbf{x}, \quad \mathbf{u}_0 = \mathbf{x}. \tag{2.2}$$

Here, u_i represents the layer activations, $\boldsymbol{\theta} = \{\mathbf{L}_i^+, \mathbf{K}_i, \mathbf{b}_i\}$ denotes the parameters or weights, and σ_i are non-linear activation functions. To ensure convexity, it is necessary to constrain the parameters $\mathbf{L}_i^+$ to be non-negative and the activation functions to be convex and non-decreasing, as demonstrated in [1] and [2]. The corresponding gradient and Hessian can be expressed as following:

$$\nabla \Phi_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla \mathbf{u}_{l-1}(\mathbf{x}) \mathbf{L}_l^{+\top} + \mathbf{K}_l^\top$$

$$\nabla^2 \Phi_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla^2 \mathbf{u}_{l-1}(\mathbf{x}) \mathbf{L}_l^{+\top}$$

which are both linear in the last layer's weights.

Since [7] leverages the gradient of the ICNNs to represent each transformation, the resultant mapping is a composition of gradients of ICNNs. Therefore, we can only utilize the linearity of the weights in the last layer of the last ICNN to the entire training process. In this project, we will discuss the efficient training method for one ICNN whose gradient represents the differomophic mapping.

To learn the weights $\boldsymbol{\theta}$ of ICNNs, we adopt the maximum likelihood training scheme. Given the assumption of normalizing flows following a diffeo-

morphic function, we can use the change of variable formula to approximate the likelihood of a given data point $\mathbf{x}$ from the target distribution $\mathcal{X}$[14]:

$$\begin{aligned}\rho_{\mathcal{X}}(\mathbf{x}) \approx \rho_{\boldsymbol{\theta}}(\mathbf{x}) &= \rho_{\eta}\left(g_{\boldsymbol{\theta}}^{-1}(\mathbf{x})\right) \det \nabla g_{\boldsymbol{\theta}}^{-1}(\mathbf{x}) \\ &= (2\pi)^{-\frac{n}{2}} \exp(-\frac{1}{2}||g_{\boldsymbol{\theta}}^{-1}(\mathbf{x})||^2) \det \nabla g_{\boldsymbol{\theta}}^{-1}(\mathbf{x}).\end{aligned}$$

Here, $\rho_{\mathcal{X}}$ denotes the true target density, $\rho_{\boldsymbol{\theta}}$ is the approximate target density, and ρ_{η} is the reference density. A common practice for utilizing this approximation to train optimal weights is by minimizing the negative log-likelihood:

$$\begin{aligned}\mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[-\log \rho_{\mathcal{X}}(\mathbf{x})\right] &\approx \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[-\log \rho_{\boldsymbol{\theta}}(\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[-\frac{n}{2}\log(2\pi) + \frac{1}{2}||g_{\boldsymbol{\theta}}^{-1}(\mathbf{x})||^2 - \log(\left|\det \nabla g_{\boldsymbol{\theta}}^{-1}(\mathbf{x})\right|)\right].\end{aligned}$$

Since the first term of the negative log-likelihood is a constant, we can ignore it and focus on minimizing the second and third terms:

$$\arg\min_{\boldsymbol{\theta}} \quad \mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\frac{1}{2}||g_{\boldsymbol{\theta}}^{-1}(\mathbf{x})||^2 - \log(\left|\det \nabla g_{\boldsymbol{\theta}}^{-1}(\mathbf{x})\right|)\right].$$

This is also equivalent to minimizing the Kullback-Leibler (KL) divergence between the true density and modeled density:

$$D_{\mathrm{KL}}\left[\rho_{\mathcal{X}} \parallel \rho_{\boldsymbol{\theta}}\right] := \int_{\mathbb{R}^n} \rho_{\mathcal{X}}(\mathbf{x}) \log\left(\frac{\rho_{\mathcal{X}}(\mathbf{x})}{\rho_{\theta}(\mathbf{x})}\right) d\mathbf{x} = \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\log\left(\frac{\rho_{\mathcal{X}}(\mathbf{x})}{\rho_{\boldsymbol{\theta}}(\mathbf{x})}\right)\right].$$

Since the density $\rho_{\mathcal{X}}(\mathbf{x})$ is unknown and independent of our model, we aim to minimize the other term which corresponds exactly to the negative log-likelihood.

For gradient of input convex neural networks as the inverse of the generator specifically, we have the following loss function:

$$\mathcal{J}[\Phi_{\boldsymbol{\theta}}] = \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\frac{1}{2}\|\nabla\Phi_{\boldsymbol{\theta}}(\mathbf{x})\|^2 - \log\det\nabla^2\Phi_{\boldsymbol{\theta}}(\mathbf{x})\right]. \tag{2.3}$$

Stochastic gradient descent method, and its variants, Adam [8], are commonly used for weights training. A key observation here is the objective function is convex in $\Phi_{\boldsymbol{\theta}}$, since both $\nabla\Phi_{\boldsymbol{\theta}}(\mathbf{x})$ and $\nabla^2\Phi_{\boldsymbol{\theta}}(\mathbf{x})$ are linear transformation of $\Phi_{\boldsymbol{\theta}}$, and the square norm of a convex function preserve convexity and the log-determinant of a matrix is concave. Therefore, the addition between two convex function presevre convexity. The non-convexity of the loss function $\mathcal{J}$ in the weights $\boldsymbol{\mu}$ of all layers except for the last presents a significant challenge. This is due to the nonlinear nature of ICNNs and the fact that preservation of convexity is only guaranteed for weights that enter in an affine manner. Nevertheless, since the last layer of ICNNs is indeed an affine mapping that preserve convexity proved by [2], we could exploit this fact to make the training more efficient.

2.2 Variable Projection

Variable projection (VarPro) is a method initially developed for solving nonlinear separable least square problems, as defined by Golub and Pereyra [6]. The key idea here is to eliminate the linear parameters in the objective function by solving a convex optimization problem. For details, see [5]. This method offers a significant advantage in terms of convergence since it requires fewer iterations to converge than the minimization of the full objective problem due to the better-conditioned reduced problem. Numerical experiments have been conducted to verify the efficacy of this approach [13].

Both [12] and [10] have explored the use of VarPro for training deep neural networks (DNNs) on regression and classification tasks. Since most of state-of-art architecture of DNNs' last layers are affine mapping, the main idea is to use VarPro to obtain a reduced loss function from the original loss function. By solving an optimization problem for linear weights in the last layer in terms of function of the other nonlinear weights, we implicitly account for the coupling between the linear weights and the nonlinear weights when eliminating the linear parameters from the loss function, which leads to a faster convergence as illustrate by Figure 2.1.

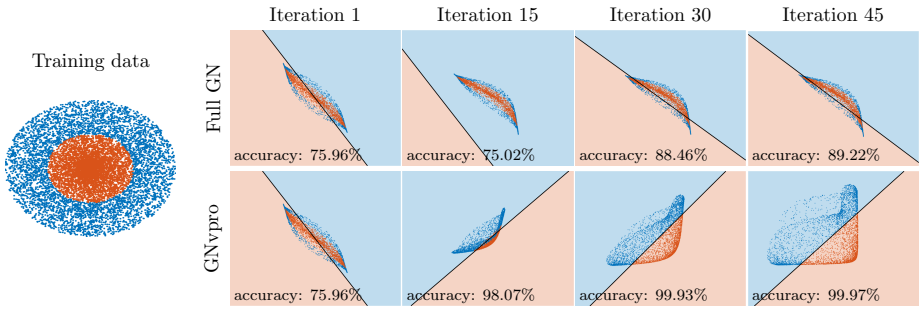

Figure 2.1: Binary classification problem trust region Gauss-Newton-Krylov and Tikhonov regularization with and without VarPro [12]. We observe that Gauss-Newton-Krylov with VarPro (GNvpro) completes the classification task with fewer iteration and higher accuracy than Gauss-Newton-Krylov without VarPro (Full GN) because GNvpro implicitly accounts for coupling.

Chapter 3

Efficient Method in Training ICNNs

In this chapter, we introduce the numerical efficient approach for training ICNNs. Motivated by existing work of VarPro training for least square loss and cross-entropy loss, we employed Varpro training scheme for negative log-likelihood loss. We begin by approximating the expectation by sample average and rewriting the loss function in terms of linear weights and nonlinear weights, followed by detailed explanation of how to solve the inner and outer problems for training ICNNs. Finally, we will present the implementation of the training schemes.

3.1 Sample Average Approximation

In this work, we leverage the separable structure of ICNNs as defined in (2.1),

$$\Phi_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{u}_l = \mathbf{L}_l^+ \mathbf{u}_{l-1} + \mathbf{K}_l \mathbf{x}$$

where the last layer is an affine mapping:

$$\Phi_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{w}^\top f_{\boldsymbol{\mu}}(\mathbf{x}), \tag{3.1}$$

Here, $f_{\boldsymbol{\mu}} = [\mathbf{u}_{l-1}, \mathbf{x}]^T : \mathbb{R}^n \to \mathbb{R}^m$ denotes all but the last ICNN layer, m is the width or the number of neurons of the penultimate layer, $\boldsymbol{\mu}$ are the weights of all but the last layers, i.e. $\{\mathbf{L}_1^+, \mathbf{K}_1, \mathbf{b}_1, ..., \mathbf{L}_{l-1}^+, \mathbf{K}_{l-1}, \mathbf{b}_{l-1}\}$, and $\mathbf{w} \in \mathbb{R}^m$ are the weights of the last layer, i.e. $[\mathbf{L}_l^+, \mathbf{K}_l]^T$. Notably, the model is linear in $\mathbf{w}$, and thus, the objective function is convex in $\mathbf{w}$. However, to preserve the convexity, all components in $\mathbf{w}$ except those mapping input path to output are constrained to be non-negative, which follows the structure of the layer of ICNNs. The nonegativity constraints defined the feasible set of $\mathbf{w}$, which will be introduced when training ICNNs.

To approximate the expected value in the loss function in (2.3), we use sample average approximation (SAA) that leads to a deterministic optimization problem that using large batches of randomly chosen sample N (sufficient

large sample to avoid overfitting):

$$\mathcal{J}[\Phi_{\boldsymbol{\theta}}] \approx J[\Phi_{\boldsymbol{\theta}}] = \frac{1}{N}\sum_{j=1}^{N}\left[\frac{1}{2}\|\nabla\Phi_{\boldsymbol{\theta}}(\mathbf{x}_j)\|^2 - \log\det\nabla^2\Phi_{\boldsymbol{\theta}}(\mathbf{x}_j)\right],$$

and then minimize J with respect to the parameters of Φ.

Given the separability of ICNNs (3.1), we can rewrite the loss function as:

$$J(\boldsymbol{\mu}, \mathbf{w}) = \frac{1}{N}\sum_{j=1}^{N}\left[\frac{1}{2}\|\nabla f_{\boldsymbol{\mu}}(\mathbf{x}_j)\mathbf{w}\|^2 - \log\det\sum_{i=1}^{m}\mathbf{w}_i\nabla^2 f_{\boldsymbol{\mu}}^{(i)}(\mathbf{x}_j)\right]$$

where $f_{\boldsymbol{\mu}}^{(i)}$ denotes the ith component of $f_{\boldsymbol{\mu}}$.

3.2 Training ICNNs with Variable Projection

We divided the training problem of ICNNs into the inner problem:

$$\mathbf{w}(\boldsymbol{\mu}) = \arg\min_{\mathbf{w}\in\mathcal{A}} J(\boldsymbol{\mu}, \mathbf{w}), \text{ where } \mathcal{A} \text{ is the feasible set (explained in 3.1)}$$

and reduced problem:

$$J_{\text{red}}(\boldsymbol{\mu}) = J(\boldsymbol{\mu}, \mathbf{w}(\boldsymbol{\mu}))$$

according to the VarPro training scheme.

The inner problem is indeed a constraint convex optimization problem:

$$\min J(\boldsymbol{\mu}, \mathbf{w}), \text{ s.t. } w_i \geq 0 \text{ for } i = 1, 2, \ldots, m-n.$$

Here, we use log barrier method to reformulate the objective function via an indicator function:

$$J(\boldsymbol{\mu}, \mathbf{w}) + \sum_{i=1}^{m-n} I_{-}(w_i),$$

where $I_{-}(\mathbf{u}) = 0$ if $\mathbf{u} \geq 0$, $I_{-}(\mathbf{u}) = \infty$ otherwise.

Here, the indicator function is approximated by logarithmic barrier:

$$J_{\log}(\boldsymbol{\mu}, \mathbf{w}) = J(\boldsymbol{\mu}, \mathbf{w}) - \frac{1}{t_0} \sum_{i=1}^{m-n} \log(w_i),$$

and the approximation improves as $t_0 \to \infty$ since the duality gap is defined by m/t_0 [2]. This approximation allows us to have twice differentiable loss function whose gradient and Hessian are:

$$\nabla J_{\log}(\boldsymbol{\mu}, \mathbf{w}) = \nabla J(\boldsymbol{\mu}, \mathbf{w}) - \frac{1}{t_0} \left[\begin{array}{c} w_1^{-1}, \ldots, w_{m-n}^{-1}, 0, \ldots, 0 \end{array} \right]^T, \tag{3.2}$$

where last n entries of the vector in the second term is zero;

$$\nabla^2 J_{\log}(\boldsymbol{\mu}, \mathbf{w}) = \nabla^2 J(\boldsymbol{\mu}, \mathbf{w}) + \frac{1}{t_0} \begin{bmatrix} w_1^{-2} & & & & & \\ & \ddots & & & & \\ & & w_{m-n}^{-2} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} \tag{3.3}$$

Then we employ Newton's method for each centering steps and updating t_0 until the duality gap is smaller than the tolerance.

Algorithm 1 Log Barrier Method for Updating $\mathbf{w}$

given feasible initial $\mathbf{w}$, $t_0 := t_0^{(0)} > 0$, $c > 1$, duality gap tolerance ϵ_1, newton decrement tolreance ϵ_2.
repeat
1. *Centering Step and Update*:
repeat
i. *Compute Newton step and decrement.*

$$\Delta \mathbf{w}_{nt} := -\nabla^2 J_{\log}(\boldsymbol{\mu}, \mathbf{w})^{-1} \nabla J_{\log}(\boldsymbol{\mu}, \mathbf{w})$$
$$\lambda^2 := -\nabla J_{\log}(\boldsymbol{\mu}, \mathbf{w})^T \nabla \mathbf{w}_{nt}$$

ii. *Stopping criterion for newton.* **quit** with $\mathbf{w}^*(t_0)$ if $\lambda^2/2 < \epsilon_2$
iii. *Line search.* Choose step size t by backtracking line search.
iv. *Update.* $\mathbf{w} := \mathbf{w} + t\Delta \mathbf{w}_{nt}$
Obtain updated $\mathbf{w} := \mathbf{w}^*(t_0)$
2. *Stopping criterion for barrier method*: **quit** if $m/t_0 < \epsilon_1$
3. *Shrinking duality gap by updating* t_0: $t_0 := ct_0$

Then, after we obtain the reduced optimization problem, the nonlinear

weights can be updated in different ways, e.g., using gradient descent or ℓBFGS. Noted here, the gradient of the reduced objective function is:

$$\nabla_{\boldsymbol{\mu}} J_{\text{red}}(\boldsymbol{\mu}) = \mathbf{J}_{\boldsymbol{\mu}}\mathbf{w}(\boldsymbol{\mu})^T \nabla_{\mathbf{w}} J(\boldsymbol{\mu}, \mathbf{w}(\boldsymbol{\mu})) + \nabla_{\boldsymbol{\mu}} J(\boldsymbol{\mu}, \mathbf{w}(\boldsymbol{\mu})) \tag{3.4}$$

where $\mathbf{J}_{\boldsymbol{\mu}}\mathbf{w}(\boldsymbol{\mu})$ is the Jacobian of $\mathbf{w}(\boldsymbol{\mu})$. In previous work using VarPro in training DNNs by [12, 10], the optimality condition of the inner problem is:

$$\nabla_{\mathbf{w}} J(\boldsymbol{\mu}, \mathbf{w}(\boldsymbol{\mu})) = 0$$

Then, they updated the nonlinear weights without tracking the gradient of the inner problem. Nevertheless, since we use interior point method for solving the inner problem the gradient may not be zero for optimality condition when the solution is close to the boundary [2].

However, when we perform the gradient check using Taylor Remainder, the gradient of the reduced problem is still accurate even when we do not track the gradient of the inner problem. Here, we can formulate the first-order Taylor Remainder gradient check as the following:

$$\text{error} = |J_{\text{red}}(\boldsymbol{\mu}) + h\nabla J_{\text{red}}(\boldsymbol{\mu})^\top \Delta\boldsymbol{\mu} - J_{\text{red}}(\boldsymbol{\mu} + h\Delta\boldsymbol{\mu})|$$

where $\Delta\boldsymbol{\mu}$ is a small perturbation of all the network weights except for the last layer $\boldsymbol{\mu}$. We observe that the error converge like $O(h^2)$ as $h \to 0$, indicating the first term in 3.4 is still small.

h	error	ratio
1.00e+00	1.76e-02	-
5.00e-01	4.33e-03	4.06e+00
2.50e-01	1.06e-03	4.08e+00
1.25e-01	2.75e-04	3.85e+00
6.25e-02	6.68e-05	4.12e+00

Table 3.1: Taylor reminder graident check for the gradient of nonlinear weights

Here, the error converge approximately like $O(h^2)$ due to the fact that we use log barrier to approximate the constraint. We will discuss alternative method in the discussion section for solving this interior point method.

Then, we can updated the nonlinear weights with ℓBFGS without tracking the gradient when solving the inner problem.

Finally, we present the entire algorithm below:

Algorithm 2 Efficient Training ICNNs with VarPro

Inputs: A Large Batch $\mathcal{T} = \{\mathbf{x}_j : j = 1, 2, \ldots, N\}$ from distribution $\mathcal{X}$,
Forward propagate
1: **for** $j = 1, \ldots, N$ for $\mathcal{T}$ **do**
2: $\quad \Phi_{\boldsymbol{\theta}}(\mathbf{x}_k) = \mathbf{w}^T f_{\boldsymbol{\mu}}(x_k)$
3: **end for**
Evaluate $\mathcal{J}_c$ with SAA and applying VarPro
4: $\mathbf{w}(\boldsymbol{\mu}) = \arg\min_{\mathbf{w}\in\mathcal{A}} J(\boldsymbol{\mu}, \mathbf{w})$ (Solving by Algorithm 1)
5: $\mathcal{J}_c = J_{\text{red}}(\boldsymbol{\mu}) = J(\boldsymbol{\mu}, \mathbf{w}(\boldsymbol{\mu}))$
6: $\nabla\mathcal{J}_{\text{c}} = \nabla_{\boldsymbol{\mu}} J_{\text{red}}(\boldsymbol{\mu})$
7: Optimize $\boldsymbol{\mu}$ with ℓBFGS or gradient descent method
8: **Repeat until convergence**

Chapter 4

Numerical Experiments

In this section, we demonstrate the effectiveness and efficiency of our training scheme in moons dataset (two dimension synthetic dataset). We present the baseline training scheme with Adam and then compare it with variable projection training schemes. We perform the experiment on Google Colab with Python 3 Google Compute Engine backend, a part of Google Cloud Platform (GCP), which provides scalable and customizable virtual machines to meet various computational requirements. We largely utilize the pytorch optimization package in updating weights with Adam and ℓBFGs optimizer.

Experiments with the two dimension AI synthesis dataset Moons show that variable projection training scheme is more efficient in updating weights of ICNNs in terms of achieving a lower test loss and fewer gradient evaluation than the basline training method with Adam.

4.1 Experiment with Moons Dataset

We consider test loss and number of gradient evaluation as the two primary metrics. We define the gradient evaluation of a single forward propagation and backward propagation for one sample as one work unit. Although training with variable projection (VarPro) necessitates additional computational cost for solving the inner problem, this extra cost is negligible compared to the cost of one work unit. We also leverage the fast computation of Hessian matrices in [11] for VarPro training.

We use the stochastic approximation and updating all weights of ICNNs using Adam as our baseline. The ICNN used for experiment has 4 hidden layers and 32 neurons per hidden layer. We use the PyTorch Adam optimizer with learning rate = 0.05. We use a mini-batch with size 64. We then employ the VarPro training schemes and updating the nonlinear weights with Adam (Adam VarPro) and ℓBFGs (ℓBFGs VarPro) for the same initialization of weights. For sample average approximation, we use a large batch with size 512 for Adam VarPro and size 1024 for ℓBFGs VarPro.

We found that the convergence of ℓBFGs largely depends on initialization of weights. This is expected since ℓBFGs, like other quasi-Newton methods, is sensitive to the initilization [9]. Additionally, Adam VarPro resulting results in higher training and validation loss at convergence since this do not follow the principle of sample average approximation and the expectation in 2.3 is intractable. However, this method serve as a good warm up for initialization,

which results in our preferable training scheme Adam-ℓBFGs VarPro.

	Training	Validation	Test
Full Adam	-1.15 ± 0.07	-1.12 ± 0.04	-1.13 ± 0.03
Adam VarPro	-1.04 ± 0.14	-0.98 ± 0.11	-0.96 ± 0.13
Adam-ℓBFGs VarPro	-1.35 ± 0.04	-1.33 ± 0.06	-1.32 ± 0.05

Table 4.1: Training, validation, and test results for training ICNNs. The sample size for both validation and test are 5000. We report mean loss and standard deviation where the smaller values indicate better fit.

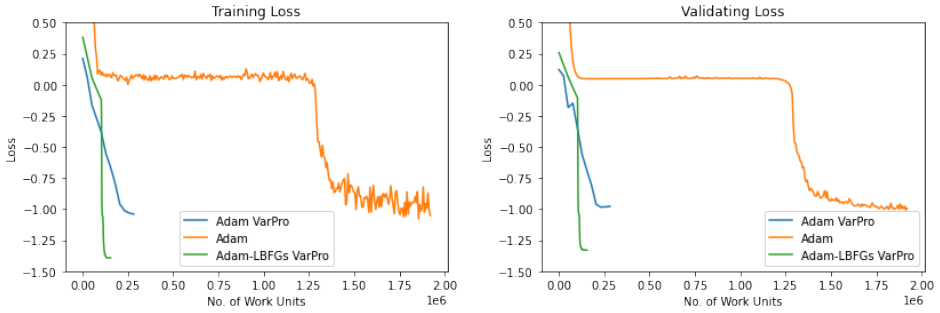

Figure 4.1: We plot the convergence of the training and validation loss among three methods versus number of working units. Adam VarPro and Adam-ℓBFGs VarPro both converge faster than Full Adam with less working units, but Adam-ℓBFGs VarPro converge to lower training loss.

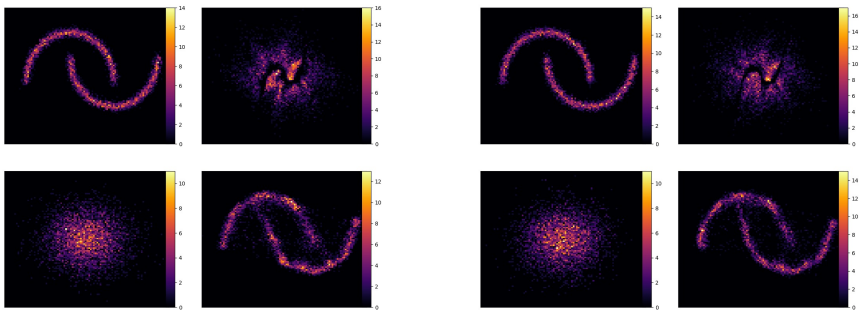

Figure 4.2: Left: Training with Full Adam. **Right**: Training with Adam-ℓBFGs VarPro. For each graph, **Top Left**: samples from the target distribution. **Top Right**: distribution of the inverse map outputs. **Bottom Left**: samples from the reference distribution. **Bottom Right**: generated samples.

Chapter 5

Discussion

The numerical results presented in Figure 4.1 suggest that training ICNNs with variable projection (VarPro) results in a lower test loss and fewer gradient evaluation for training the moons dataset. However, the generated samples in Figure 4.2 imply that representing the diffeomorphic mapping by one ICNN may not be the best mapping we would like to adpot eventually. We could implement this variable projection technique to train other ICNN-based models. Applying this training scheme to original mapping described in [7] could result in faster convergence and lower test loss. Nevertheless, we can only use variable projection for the last ICNN since the original mapping is a composition of ICNNs.

In addition to original mapping, we consider to apply this training scheme to the triangular convex flows introduced recently. Since the mapping is parametrize with fully input convex neural networks and partially input con-

vex neural networks (a slightly different architecture which will results in a more complicated inner problem), we could potentially apply the VarPro training scheme to both networks to reach a better result.

Finally, other alternative constraint optimization method like primal-dual interior point method [2] could be also applied to solve the inner problem more efficiently.

Chapter 6

Conclusion

We train an ICNN efficiently with variable projection (VarPro). This method exploits the fact of the affine mapping in the last layer of ICNNs. Additionally, VarPro training scheme implicitly account for coupling between linear weights and nonlinear weights leading to a lower test loss. Numerical experiments demonstrate the effectiveness and efficiency of VarPro training scheme. Future directions including using more efficient and effective method in solving the inner problems and applying this training schemes to partial input convex neural networks, and other maps that parametrize with ICNNs.

Bibliography

[1] B. Amos, L. Xu, and J. Z. Kolter. Input Convex Neural Networks. *arXiv.org*, cs.LG, 09 2016. 7

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 7, 9, 15, 17, 24

[3] Y.-J. Cao, L.-L. Jia, Y.-X. Chen, N. Lin, C. Yang, B. Zhang, Z. Liu, X.-X. Li, and H.-H. Dai. Recent Advances of Generative Adversarial Networks in Computer Vision. *IEEE Access*, 7:14985–15006, Dec. 2018. 1

[4] I. Demir and U. A. Ciftci. Where Do Deep Fakes Look? Synthetic Face Detection via Gaze Tracking. *arXiv*, Jan. 2021. 1

[5] G. Golub and V. Pereyra. Separable nonlinear least squares: the variable projection method and its. *Inverse Prob.*, 19(2):R1, Feb. 2003. 10

[6] G. H. Golub and V. Pereyra. The Differentiation of Pseudo-Inverses and Nonlinear Least Squares Problems Whose Variables Separate on JSTOR. *SIAM J. Numer. Anal.*, 10(2):413–432, Apr. 1973. 10

[7] C.-W. Huang, R. T. Q. Chen, C. Tsirigotis, and A. Courville. Convex Potential Flows: Universal Probability Distributions with Optimal Transport and Convex Optimization. *arXiv*, cs.LG, 12 2020. 2, 5, 6, 7, 23

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. 9

[9] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45(1):503–528, Aug. 1989. 20

[10] E. Newman, J. Chung, M. Chung, and L. Ruthotto. slimTrain – A Stochastic Approximation Method for Training Separable Deep Neural Networks. *arXiv*, 2021. 3, 10, 17

[11] E. Newman and L. Ruthotto. hessQuik: Fast Hessian computation of composite functions. *Journal of Open Source Software*, 7(72):4171, 2022. 20

[12] E. Newman, L. Ruthotto, J. Hart, and B. v. B. Waanders. Train Like a (Var)Pro: Efficient Training of Neural Networks with Variable Projection. *arXiv*, cs.LG, 07 2020. 3, 10, 11, 17

[13] D. M. O'Leary and B. W. Rust. Variable Projection for Nonlinear Least Squares Problems. *NIST*, Aug. 2012. 10

[14] L. Ruthotto and E. Haber. An Introduction to Deep Generative Modeling. *GAMM Mitteilungen*, cs.LG, 03 2021. 2, 6, 8

[15] S. Wang, Y. Yang, Z. Wu, Y. Qian, and K. Yu. Data Augmentation Using Deep Generative Models for Embedding Based Speaker Recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.*, 28:2598–2609, Aug. 2020. 1