Modeling Structured Data with Invertible Generative Models

You Lu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Application

Bert Huang, Co-chair
Naren Ramakrishnan, Co-chair
Jia-Bin Huang
Anuj Karpatne
Junier Oliva

December 17, 2021
Blacksburg, Virginia

Keywords: Machine Learning, Computer Vision, Deep Generative Models, Structured Prediction, Weakly Supervised Learning, Normalizing Flows,

Modeling Structured Data with Invertible Generative Models

You Lu

(ABSTRACT)

Data is complex and has a variety of structures and formats. Modeling datasets is a core problem in modern artificial intelligence. Generative models are machine learning models, which model datasets with probability distributions. Deep generative models combine deep learning with probability theory, so that can model complicated datasets with flexible models. They have become one of the most popular models in machine learning, and have been applied to many problems.

Normalizing flows are a novel class of deep generative models that allow efficient exact likelihood calculation, exact latent variable inference and sampling. They are constructed using functions whose inverse and Jacobian determinant can be efficiently computed. In this paper, we develop normalizing flow based generative models to model complex datasets. In general, data can be categorized to unlabeled data, labeled data, and weakly labeled data. We develop models for these three types of data, respectively.

First, we develop *Woodbury transformations*, which are flow layers for general unsupervised normalizing flows, and can improve the flexibility and scalability of current flow based models. Woodbury transformations achieve efficient invertibility via Woodbury matrix identity and efficient determinant calculation via Sylvester's determinant identity. In contrast with other operations used in state-of-the-art normalizing flows, Woodbury transformations enable (1) high-dimensional interactions, (2) efficient sampling, and (3) efficient likelihood evaluation. Other similar operations, such as 1x1 convolutions, emerging convolutions, or periodic convolutions allow at most two of these three advantages. In our experiments on multiple

image datasets, we find that Woodbury transformations allow learning of higher-likelihood models than other flow architectures while still enjoying their efficiency advantages.

Second, we propose *conditional Glow (c-Glow)*, a conditional generative flow for structured output learning, which is an advanced variant of supervised learning with structured labels. Traditional structured prediction models try to learn a conditional likelihood, i.e., $p(\mathbf{y}|\mathbf{x})$, to capture the relationship between the structured output $\mathbf{y}$ and the input features $\mathbf{x}$. For many models, computing the likelihood is intractable. These models are therefore hard to train, requiring the use of surrogate objectives or variational inference to approximate likelihood. C-Glow benefits from the ability of flow-based models to compute $p(\mathbf{y}|\mathbf{x})$ exactly and efficiently. Learning with c-Glow does not require a surrogate objective or performing inference during training. Once trained, we can directly and efficiently generate conditional samples. We develop a sample-based prediction method, which can use this advantage to do efficient and effective inference. In our experiments, we test c-Glow on five different tasks. C-Glow outperforms the state-of-the-art baselines in some tasks and predicts comparable outputs in the other tasks. The results show that c-Glow is applicable to many different structured prediction problems.

Third, we develop label learning flows (LLF), a general framework for weakly supervised learning problems. Our method is a generative model based on normalizing flows. The main idea of LLF is to optimize the conditional likelihoods of all possible labelings of the data within a constrained space defined by weak signals. We develop a training method for LLF that trains the conditional flow inversely and avoids estimating the labels. Once a model is trained, we can make predictions with a sampling algorithm. We apply LLF to three weakly supervised learning problems. Experiment results show that our method outperforms many baselines we compare against.

Our research shows the advantages and versatility of normalizing flows.

Modeling Structured Data with Invertible Generative Models

You Lu

(GENERAL AUDIENCE ABSTRACT)

Data is now more affordable and accessible. At the same time, datasets are more and more complicated. Modeling data is a key problem in modern artificial intelligence and data analysis. Deep generative models combine deep learning and probability theory, and are now a major way to model complex datasets. In this dissertation, we focus on a novel class of deep generative model–normalizing flows. They are becoming popular because of their abilities to efficiently compute exact likelihood, infer exact latent variables, and draw samples. We develop flow-based generative models for different types of data, i.e., unlabeled data, labeled data, and weakly labeled data. First, we develop *Woodbury transformations* for unsupervised normalizing flows, which improve the flexibility and expressiveness of flow based models. Second, we develop *conditional generative flows* for an advanced supervised learning problem – structured output learning, which removes the need of approximations, and surrogate objectives in traditional (deep) structured prediction models. Third, we develop *label learning flows*, which is a general framework for weakly supervised learning problems. Our research improves the performance of normalizing flows, and extend the applications of them to many supervised and weakly supervised problems.

Dedication

This dissertation is dedicated to my beloved parents, who are the most important people in my life. This dissertation is also dedicated to my maternal grandparents and my paternal grandmother. My maternal grandparents passed away in 2018, and my paternal grandmother passed away in 2021. I hope they are at peace and happy in Heaven.

Acknowledgments

First and foremost, I would like to owe my deepest appreciation to my advisor, Dr. Bert Huang, for his advice, support, and guidance. Dr. Huang is a knowledgeable and outstanding advisor. He generously shares with me his invaluable knowledge and experience in research, and guide me to be the best researcher possible. Without his guidance, I would not complete this dissertation. I am very grateful to be one of his students. Working with him is a precious experience that I will never forget.

Second, I would like to thank other committee members, Dr. Naren Ramakrishnan, Dr. Jia-Bin Huang, Anuj Karpatne, and Dr. Junier Oliva, for accepting to serve on my committee. Their insightful comments and suggestions help me improve my dissertation.

Finally, I would like to thank all my lab mates. It was great working and studying with them in the past four years. Their help and support are meaningful to me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this era of information explosion, data is becoming more affordable and accessible. For example, downloading images from websites only requires a click of button. Copying contents from websites is free and simple. At the same time, datasets are more and more complex. Data points are becoming multimodal, high-dimensional and structured. In modern artificial intelligence, modeling complex data with machine learning methods is the key to build successful data driven intelligent agents.

In general, data can be categorized to three types, i.e., unsupervised data, supervised data and weakly supervised data. Accordingly, machine learning algorithms are grouped to unsupervised learning, supervised learning, and weakly supervised learning methods, respectively, based on what type of data they intend to model.

Unlabeled data consists of samples that are either taken from nature or created by human. Examples of them include photos, web pages, articles, etc. Unsupervised learning [30] develops models to capture patterns and properties occurring in the datasets.

Supervised data augments each unsupervised data point with meaningful labels identifying certain properties. For example, given a set of images, the label of each image can be if this image contains a dog or a cat. Supervised learning learns a map between the input data and output labels.

In many situations, data-labeling is costly. We therefore use weakly supervised data, in

which only inaccurate or incomplete supervision information is available. Weakly supervised learning [128] wants to predict labels for data points with only this weakly supervision information. For example, in weakly supervised classification, each data point only has noisy estimate of labels attained by rule-based labeling methods. We train a classifier that takes in these data points and noisy labels and predict true labels.

Generative models are powerful tools for modeling complex distributions. Classical generative models include Bayesian nets and Markov random fields [108], which represent the joint distribution of variables with a graph. Deep generative models combine deep learning with probability theory, and use neural networks to represent relationships among variables. Representatives of them include generative adversarial networks [31], variational auto-encoders [50, 93], autoregressive models [84] and normalizing flows [21, 22, 51, 92]. Generative models have the potential to capture all dependencies and patterns existing in the data, so that can learn meaningful and robust features [51]. Nowadays, they have been applied to model many different formats of data including texts [13, 20], images [31, 50], graphs [59, 122], 3d shapes [113, 117], and audios [83], etc. Practically, they have been widely used in tasks such as synthetic data generation [83, 120], domain adaption [129], density estimation [22], and structured prediction [100].

In this work, we focus on normalizing flows, which are a novel class of deep generative models. Normalizing flows have the unique advantages of exact likelihood learning and exact latent code inference, so that are getting more and more attention in recent years. We explore the use of normalizing flows to model complex and structured data. First, we develop Woodbury transformations for unsupervised normalizing flows. Second, we develop conditional generative flows(c-Glow) for structured prediction – a variant of supervised learning. Third, we develop label learning flows (LLF), which is a general framework for weakly supervised learning. Detailed introduction of these works are as follows.

Woodbury Transformations. Flow layers are specific designed neural networks with tractable Jacobian determinant and inverse. To preserve this computational efficiency, these layers usually cannot sufficiently encode dependencies among dimensions of a variable. Some linear transformation layers [21, 27, 39, 51], i.e., generalized permutation layers, are developed to model the dependencies among dimensions. However, the 1×1 convolution [51] are not flexible enough, and the $d \times d$ convolutions [27, 39] do not scale well to high-dimensional variables. In this work, we develop Woodbury transformations, which use Woodbury matrix identity and Sylvester's determinant identity to efficiently compute the inverse and Jacobian determinant. Our methods improve the scalability and flexibility of normalizing flows, and can be applied to general unsupervised flow based models.

Conditional Generative Flows for Structured Prediction. Traditional structured prediction models use energy based models such as MRFs [108] to define the conditional distribution $p(\mathbf{y}|\mathbf{x})$. Training these models is difficult, and requires using surrogate objectives or approximate inference, because the likelihood is intractable. We develop conditional generative flows (c-Glow) – a model for general structured prediction problems. C-Glow benefits from normalizing flows, and can be trained directly and efficiently by optimizing the likelihood, removing the need for surrogates or inference. We develop a sample based prediction method, which uses advantage of normalizing flows, and estimate labels efficiently and effectively. Our experiments on five structured prediction tasks show that c-Glow can predict high-quality outputs.

Label Learning Flows. Many constrained based weakly supervised learning methods [4, 8, 89, 91] learn a deterministic function to estimate labels given input data and weak signals, and ignore the uncertainty between input data and output labels. *Label learning flows* (LLF) is a general framework for weakly supervised learning problems, which uses a conditional flow to model the probability of possible outputs given input data. We develop a learning

method for LLF that trains the conditional flow inversely and avoids estimating the labels. For prediction, we use the same method as c-Glow and estimate labels with sample average.

This dissertation is organized as follows. In Chapter 2, we introduce basic concepts that most related to this thesis. In Chapter 3, we introduce Woodbury transformations. In Chapter 4, we introduce c-Glow. In Chapter 5, we introduce LLF. In Chapter 6, we conclude this dissertation and discuss some open problems.

Chapter 2

Background

In this section, we introduce the basic concepts that related to this dissertation.

2.1 Unsupervised Learning

Given a collected dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_D\}$, where $\mathbf{x}_i$ is the ith data point, the goal of unsupervised learning is to develop representations from $\mathcal{D}$ that can be used for prediction, sampling, reasoning, etc [30]. In this paper, we focus on probabilistic models. Suppose that $\mathcal{D}$ is drawn from the true data distribution $p^*(\mathbf{x})$, we are interested in approximating $p^*(\mathbf{x})$ with a model $p_\theta(\mathbf{x})$. The parameter θ can be learned by minimizing the negative log-likelihood

$$\mathcal{L}(\mathcal{D}) = \sum_{i=1}^{D} -\log p_\theta(\mathbf{x}_i). \tag{2.1}$$

2.2 Structured Output Learning

Structured output learning, i.e., structured prediction, is an advanced variant of supervised learning, where the label $\mathbf{y}$ comes from a complex, high-dimensional output space $\mathcal{Y}$ with dependencies among dimensions. Given a dataset $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$, where $\mathbf{x}_i$ is the ith input and $\mathbf{y}_i$ is the corresponding output, we suppose that the relationship between $\mathbf{x}$ and $\mathbf{y}$ is defined by an unknown true distribution $p^*(\mathbf{y}|\mathbf{x})$. We approximate the true

distribution with a model $p(\mathbf{y}|\mathbf{x}, \theta)$, which can be trained by minimizing the negative log-likelihood

$$\mathcal{L}(\mathcal{D}) = -\frac{1}{N}\sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x}_i, \theta). \tag{2.2}$$

Many structured output learning approaches use an energy-based model [62] to define this conditional distribution:

$$p(\mathbf{y}|\mathbf{x}) = \frac{e^{E(\mathbf{y},\mathbf{x})}}{\int_{\mathbf{y}'\in\mathcal{Y}} e^{E(\mathbf{y}',\mathbf{x})} d\mathbf{y}}, \tag{2.3}$$

where $E(.,.) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}$ is the energy function. In deep structured prediction, $E(\mathbf{x}, \mathbf{y})$ depends on $\mathbf{x}$ via a deep network. Due to the high dimensionality of $\mathbf{y}$, the partition function, i.e., $\int_{\mathbf{y}'\in\mathcal{Y}} e^{E(\mathbf{y}',\mathbf{x})} d\mathbf{y}$, is intractable. To train this model, we need methods to approximate the partition function such as variational inference or surrogate objectives, resulting in complicated training and sub-optimal results.

2.3 Weakly Supervised Learning

Given a dataset $\mathcal{D} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, and weak signals $\mathbf{Q}$, weakly supervised learning finds a model that can predict the unknown label $\mathbf{y}_i$ for each input data $\mathbf{x}_i$. Existing constrained-based methods, e.g., ALL [4], define a set of constrained functions based on $\mathbf{Q}$ and $\mathbf{x}$. These functions form a space of possible $\mathbf{y}$. The methods then estimate one $\mathbf{y}$ within this constrained space. However, this constrained space may be loose, and there can be more than one possible $\mathbf{y}$s satisfying these constraints. These methods ignore this uncertainty between $\mathbf{x}$ and $\mathbf{y}$.

2.4 Classical Generative Models

Before deep learning thrives, generative models are developed purely based on probability theory. These models are essentially hierarchical and structured distributions of a set of random variables. In general, we can represent a model with a graph $G = (V, E)$, where $V = \{1, ..., n\}$ be a set of n vertices, and E be a set of edges. Each vertex i in the graph corresponds to a random variable $\mathbf{x}_i$, and each edge (i, j) represents the dependency between two variables $\mathbf{x}_i$, $\mathbf{x}_j$. Therefore, these models are also called graphical models [108].

Bayesian nets are directed graphical models. An edge (i, j) indicates $\mathbf{x}_j$ is conditioned on $\mathbf{x}_i$. Given a Bayesian net, the joint distribution of these n variables can be factorized as follows:

$$p(\mathbf{x}_1, ..., \mathbf{x}_n) = \prod_{i=1}^{n} p(\mathbf{x}_i | \pi(\mathbf{x}_i)),$$

where $\pi(\mathbf{x}_i)$ is the parent set of $\mathbf{x}_i$.

Markov random fields (MRFs) are undirected graphical models. Given an MRF, the joint distribution of variables are factorized according to cliques of the graph. Let $\mathcal{C}$ be a set of all cliques in G, the $C \in \mathcal{C}$, and $\phi_C(\mathbf{x}_C) : \prod_{i \in C} \mathcal{X}_i \to R$, be the potential function of variables in C, the distribution of $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$ can be written as:

$$p(\mathbf{x}_1, ..., \mathbf{x}_n) = \frac{1}{Z} \prod_{C \in \mathcal{C}} e^{\phi_C(\mathbf{x}_C)},$$

where the Z is the partition function, and is for normalizing the distribution.

Graphical models explicitly define the joint distribution of variables. One essential drawback of them is the model complexity increases with the model size. Small models are not able to sufficiently represent the patterns and relationships among variables. However, big models will be complicated and hard to train.

2.5 Deep Generative Models

Deep generative models combine deep learning with probability theory. They implicitly define relationships among variables or probability distributions with deep neural networks, so that are more powerful and flexible, and can better model multimodal and complex data. Currently, researchers have developed various deep generative models. In this section, we introduce three representatives of them.

Generative adversarial networks (GANs) [31] use a generator G to transform latent code $\mathbf{z}$ drawn from a known distribution $p_Z(\mathbf{z})$ to real data $\mathbf{x}$. The training of GANs can be seen as an adversarial game, which uses a discriminator D to differentiate the the real data from the synthetic data generated by the generator.

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p^*(\mathbf{x})} \left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p_Z(\mathbf{z})} \left[\log(1 - D(G(\mathbf{z})))\right].$$

One downside of GANs is that they are difficult to train, and easy to suffer from the mode collapse problem, i.e., the generator generates a single sample without much variation. Many methods [6, 76, 88, 124] have been developed to improve the performance of GANs. Besides, measuring a model performance is difficult. Currently, two mainly used metrics for assessing GANs are the FID score [36] and the inception score [97]. However, these metrics also have drawbacks.

Variational auto-encoders (VAEs) [50, 93] are essentially Bayesian nets with latent variables. They assume that $\mathbf{x}$ is drawn by a posterior distribution $p(\mathbf{x}|\mathbf{z})$, where $\mathbf{z}$ is the latent variable drawn from a Gaussian prior $p_Z(\mathbf{z})$. The training of VAEs maximizes a lower bound of the likelihood, i.e., the evidence lower bound (ELBO), computed by using

variational inference. Let the variational distribution be $q(\mathbf{z}|\mathbf{x})$, the ELBO is defined as

$$\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z})\right] - KL(q(\mathbf{z}|\mathbf{x})||p_Z(\mathbf{z})).$$

Different from traditional Bayesian nets introduced in Section 2.4, VAEs use two parametric functions, e.g., neural networks, to represent the variational distribution, i.e., $q(\mathbf{z}|\mathbf{x})$, and the conditional likelihood, i.e., $p(\mathbf{x}|\mathbf{z})$, respectively, resulting in an autoencoder architecture. Some recent works develop many variants of VAE for structured data [44, 45, 53, 64, 100].

Denoising diffusion probabilistic models (DDPMs) [38] is composed of a sequence of stochastic transformations. Let $\mathbf{x}_0$ be the data, and $\mathbf{x}_1, ..., \mathbf{x}_T$ be latent states. DDPMs contain two processes. The denoising process is a Markov chain and is defined as

$$p(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p(\mathbf{x}_{t-1}|\mathbf{x}_t),$$

where $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is a conditional Gaussian distribution, and $p(\mathbf{x}_T)$ is the prior.

The diffusion process is another Markov chain and models

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}),$$

where $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is a conditional Gaussian.

The diffusion process gradually adds a small amount of Gaussian noise to the current variable at each step, and finally transform the input data $\mathbf{x}_0$ to its latent code $\mathbf{x}_T$. In sampling, we first sample a latent code $\mathbf{x}_T$ from the prior, and then gradually remove its noise with the denoising process, and finally transform it to a sample of $\mathbf{x}_0$.

2.6 Normalizing Flows

Let $\mathbf{x}$ be a high-dimensional continuous variable. In unsupervised learning, we want to train a model $p_\theta(\mathbf{x})$ by minimizing the negative log-likelihood, i.e., Equation 2.1.

Normalizing flows enable computation of $p_\theta(\mathbf{x})$, even though it is usually intractable for many other model families. A normalizing flow [92] is composed of a series of invertible functions $\mathbf{f} = \mathbf{f}_1 \circ \mathbf{f}_2 \circ ... \circ \mathbf{f}_K$, which transform $\mathbf{x}$ to a latent code $\mathbf{z}$ drawn from a simple distribution. Therefore, with the *change of variables* formula, we can rewrite $\log p_\theta(\mathbf{x})$ to be

$$\log p_\theta(\mathbf{x}) = \log p_Z(\mathbf{z}) + \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{f}_i}{\partial \mathbf{r}_{i-1}} \right) \right|, \tag{2.4}$$

where $\mathbf{r}_i = \mathbf{f}_i(\mathbf{r}_{i-1})$, $\mathbf{r}_0 = \mathbf{x}$, and $\mathbf{r}_K = \mathbf{z}$.

For some settings, variable $\tilde{\mathbf{x}}$ is discrete, e.g., image pixel values are often integers. In these cases, we dequantize $\tilde{\mathbf{x}}$ by adding continuous noise $\boldsymbol{\mu}$ to it, resulting in a continuous variable $\mathbf{x} = \tilde{\mathbf{x}} + \boldsymbol{\mu}$. As shown by Ho et al. [37], the log-likelihood of $\tilde{\mathbf{x}}$ is lower-bounded by the log-likelihood of $\mathbf{x}$.

Conditional normalizing flows [105] model the conditional distribution $p(\mathbf{y}|\mathbf{x})$. We rewrite each function as $\mathbf{f}_i = \mathbf{f}_{\mathbf{x},\phi_i}$, making it parameterized by both $\mathbf{x}$ and its parameter ϕ_i. Thus, we can rewrite the conditional likelihood as

$$\log p(\mathbf{y}|\mathbf{x}, \theta) = \log p_Z(\mathbf{z}) + \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{f}_{\mathbf{x},\phi_i}}{\partial \mathbf{r}_{i-1}} \right) \right|, \tag{2.5}$$

where $\mathbf{r}_i = \mathbf{f}_{\mathbf{x},\phi_i}(\mathbf{r}_{i-1})$.

Flow based generative models [21, 22, 51] are developed on the theory of normalizing flows. Each transformation function used in the models is a specifically designed neural network

that has a tractable Jacobian determinant and inverse. We can draw samples from a trained flow $\mathbf{f}$ by first sampling latent code $\mathbf{z} \sim p_Z(\mathbf{z})$, and then computing $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{z})$.

Compared with other (deep) generative models, normalizing flows have many advantages. They are more flexible than traditional graphical models, and can model more complex distributions without restricted assumptions. They can compute the likelihood exactly without using variational inference (VAE, DDPM) or min-max optimization (GAN), so that their training is more straightforward. DDPMs usually need hundreds of transformations to transform an input to its latent code, while normalizing flows need much fewer layers. Given these advantages, normalizing flows are getting popular recently.

2.7 Invertible Flow Layers

Recently, many flow layers have been developed to improve the performance of normalizing flows. In this section, we will discuss some commonly used ones, which are developed for general unsupervised flow based models. Let input variable $\mathbf{x}$ and its latent code $\mathbf{z}$ be $c \times h \times w$ tensors, where c is the channel dimension, and $h \times w$ are spatial dimensions.

Actnorm layers [51] perform per-channel affine transformations of the activations using scale and bias parameters to improve training stability and performance. The actnorm is formally expressed as

$$\mathbf{z}_{:,i,j} = \mathbf{s} \odot \mathbf{x}_{:,i,j} + \mathbf{b},$$

where the parameters $\mathbf{s}$ and $\mathbf{b}$ are $c \times 1$ vectors.

Affine coupling layers [21, 22] split the input $\mathbf{x}$ into two parts, $\mathbf{x}_a, \mathbf{x}_b$. And then fix $\mathbf{x}_a$ and force $\mathbf{x}_b$ to only relate to $\mathbf{x}_a$, so that the Jacobian is a triangular matrix. Formally, we

compute

$$\begin{aligned} \mathbf{x}_a, \mathbf{x}_b &= \text{split}(\mathbf{x}), \\ \mathbf{z}_a &= \mathbf{x}_a, \\ \mathbf{z}_b &= \mathbf{s}(\mathbf{x}_a) \odot \mathbf{x}_b + \mathbf{b}(\mathbf{x}_a), \\ \mathbf{z} &= \text{concat}(\mathbf{y}_a, \mathbf{z}_b), \end{aligned}$$

where $\mathbf{s}$ and $\mathbf{b}$ are two neural networks with $\mathbf{x}_a$ as input. The split and the concat split and concatenate the variables along the channel axis. Usually, s is restricted to be positive. An additive coupling layer is a special case when $\mathbf{s} = \mathbf{1}$.

Actnorm layers only rescale the dimensions of $\mathbf{x}$, and affine coupling layers only relate $\mathbf{x}_b$ to $\mathbf{x}_a$ but omit dependencies among different dimensions of $\mathbf{x}_b$. Thus, we need other layers to capture local dependencies among dimensions.

Invertible convolutional layers [27, 39, 51] are generalized permutation layers that can capture correlations among dimensions. The 1×1 convolution [51] is

$$\mathbf{z}_{:,i,j} = \mathbf{M}\mathbf{x}_{:,i,j},$$

where $\mathbf{M}$ is a $c \times c$ matrix. The Jacobian of a 1×1 convolution is a block diagonal matrix, so that its log-determinant is $hw \log |\det(\mathbf{M})|$. Note that 1×1 convolution only operates along the channel axis and ignores the dependencies along the spatial axes. To model spatial correlations, we usually use auxiliary layers to connect pixels together along channel axis.

2.7. Invertible Flow Layers

Emerging convolutions [39] combine two autoregressive convolutions [29, 52]. Formally,

$$\begin{aligned} \mathbf{M}'_1 &= \mathbf{M}_1 \odot \mathbf{A}_1, \\ \mathbf{M}'_2 &= \mathbf{M}_2 \odot \mathbf{A}_2, \\ \mathbf{z} &= \mathbf{M}'_2 \star (\mathbf{M}'_1 \star \mathbf{x}), \end{aligned}$$

where $\mathbf{M}_1, \mathbf{M}_2$ are convolutional kernels whose size is $c \times c \times d \times d$, and $\mathbf{A}_1, \mathbf{A}_2$ are binary masks. The symbol $\star$ represents the convolution operator.[1] An emerging convolutional layer has the same receptive fields as standard convolutional layers, which can capture correlations between a target pixel and its neighbor pixels. However, like other autoregressive convolutions, computing the inverse of an emerging convolution requires sequentially traversing each dimension of input, so its computation is not parallelizable and is a computational bottleneck when the input is high-dimensional.

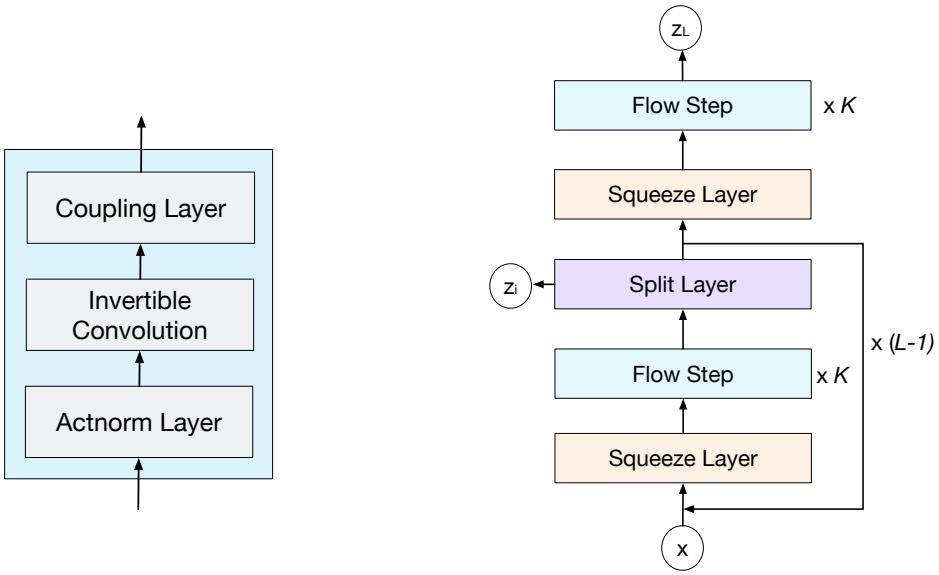
Periodic convolutions [27, 39] use discrete Fourier transformations to transform both the input and the kernel to Fourier domain. A periodic convolution is computed as

$$\mathbf{z}_{u,:,:} = \sum_v \mathcal{F}^{-1}(\mathcal{F}(\mathbf{M}^{(p)}_{u,v,:,:}) \odot \mathcal{F}(\mathbf{x}_{v,:,:})),$$

where $\mathcal{F}$ is a discrete Fourier transformation, and $\mathbf{M}^{(p)}$ is the convolution kernel whose size is $c \times c \times d \times d$. The computational complexity of periodic convolutions is $\mathcal{O}(c^2 hw \log(hw) + c^3 hw)$. In our experiments, we found that the Fourier transformation requires a large amount of memory. These two problems impact the efficiency of both training and sampling when the input is high-dimensional.

Multi-scale architectures [22, 51] compose flow layers to generate rich models, i.e., Fig-

[1] In practice, a convolutional layer is usually implemented as an aggregation of cross-correlations. We follow Hoogeboom et al. [39] and omit this detail.



(a) Flow step (b) Multi-scale architecture

Figure 2.1: Overview of architecture of generative flows. We can design the flow step by selecting a suitable convolutional layer and a coupling layer based on the task. Glow [51] uses 1×1 convolutions and affine coupling.

ure 2.1. Each flow step contains three layers, i.e., an actnorm layer, and invertible convolution layer and a coupling layer. Each level contains K flow steps, a *split layer* and a *squeeze layer*. A split layer factors out half of the variables along channel axis, and directly models them as Gaussian, so that can effectively reduce the model size. A squeeze layer divides the input $\mathbf{x}$ to subsquares of size $c \times 2 \times 2$, and then reshape them to $4c \times 1 \times 1$ subsquares. This operation transforms an $c \times h \times w$ variable to a $4c \times \frac{h}{2} \times \frac{h}{2}$ variable. Squeeze layers are used along with other invertible convolution layers to enhance correlations among dimensions.

Chapter 3

Woodbury Transformations for Deep Generative Flows

When applying flow-based models to unsupervised learning problems, we usually need to model high-dimensional variables, e.g., high-resolution images. In order to preserve computation efficiency, current flow layers usually cannot sufficiently encode dependencies among dimensions of a variable. For example, affine coupling layers [21] split a variable to two parts and require the second part to only depend on the first. But they ignore the dependencies among dimensions in the second part.

To address this problem, Dinh et al. [21, 22] introduced a fixed permutation operation that reverses the ordering of the channels of pixel variables. Kingma and Dhariwal [51] introduced a 1×1 convolution, which is a generalized permutation layer, that uses a weight matrix to model the interactions among dimensions along the channel axis. Their experiments demonstrate the importance of capturing dependencies among dimensions. Relatedly, Hoogeboom et al. [39] proposed emerging convolution operations, and Hoogeboom et al. [39] and Finz et al. [27] proposed periodic convolution. These two convolution layers have $d \times d$ kernels that can model dependencies along the spatial axes in addition to the channel axis. However, the increase in representational power comes at a cost: These convolution operations do not scale well to high-dimensional variables. The emerging convolution is a combination of two autoregressive convolutions [29, 52], whose inverse is not parallelizable. To compute the

inverse or determinant of the Jacobian, the periodic convolution requires transforming the input and the convolution kernel to Fourier space. This transformation is computationally costly.

In this chapter, we introduce *Woodbury transformations* for generative flows. Our method is also a generalized permutation layer and uses spatial and channel transformations to model dependencies among dimensions along spatial and channel axes. We use the Woodbury matrix identity [111] and Sylvester's determinant identity [102] to compute the inverse and Jacobian determinant, respectively, so that both the training and sampling time complexities are linear to the input variable's size. We also develop a memory-efficient variant of the Woodbury transformation, which has the same advantage as the full transformation but uses significantly reduced memory when the variable is high-dimensional. In our experiments, we found that Woodbury transformations enable model quality comparable to many state-of-the-art flow architectures while maintaining significant efficiency advantages.

3.1 Woodbury Transformations

In this section, we introduce Woodbury transformations as an efficient means to model high-dimensional correlations.

3.1.1 Channel and Spatial Transformations

Suppose we reshape the input $\mathbf{x}$ to be a $c \times n$ matrix, where $n = hw$. Then the 1×1 convolution can be reinterpreted as a matrix transformation

$$\mathbf{z} = \mathbf{W}^{(c)}\mathbf{x}, \tag{3.1}$$

where $\mathbf{z}$ is also a $c \times n$ matrix, and $\mathbf{W}^{(c)}$ is a $c \times c$ matrix. For consistency, we will call this a channel transformation. For each column $\mathbf{x}_{:,i}$, the correlations among channels are modeled by $\mathbf{W}^{(c)}$. However, the correlation between any two rows $\mathbf{x}_{:,i}$ and $\mathbf{x}_{:,j}$ is not captured. Inspired by Eq. 3.1, we use a spatial transformation to model interactions among dimensions along the spatial axis

$$\mathbf{z} = \mathbf{x}\mathbf{W}^{(s)}, \tag{3.2}$$

where $\mathbf{W}^{(s)}$ is an $n \times n$ matrix that models the correlations of each row $\mathbf{x}_{i,:}$. Combining Equation 3.1 and Equation 3.2, we have

$$\mathbf{x}_c = \mathbf{W}^{(c)}\mathbf{x}, \tag{3.3}$$

$$\mathbf{z} = \mathbf{x}_c\mathbf{W}^{(s)}. \tag{3.4}$$

For each dimension of output $\mathbf{z}_{i,j}$, we have $\mathbf{z}_{i,j} = \sum_{v=1}^{c} \left(\sum_{u=1}^{n} \mathbf{W}^{(c)}_{i,u} \cdot \mathbf{x}_{u,v} \right) \cdot \mathbf{W}^{(s)}_{v,j}$.

Therefore, the spatial and channel transformations together can model the correlation between any pair of dimensions. However, in this preliminary form, directly using Eq. 3.4 is inefficient for large c or n. First, we would have to store two large matrices $\mathbf{W}^c$ and $\mathbf{W}^s$, so the space cost is $\mathcal{O}(c^2 + n^2)$. Second, the computational cost of Eq. 3.4 is $\mathcal{O}(c^2 n + n^2 c)$—quadratic in the input size. Third, the computational cost of the Jacobian determinant is $\mathcal{O}(c^3 + n^3)$, which is far too expensive in practice.

3.1.2 Woodbury Transformations

We solve the three scalability problems by using a low-rank factorization. Specifically, we define

$$\begin{aligned} \mathbf{W}^{(c)} &= \mathbf{I}^{(c)} + \mathbf{U}^{(c)}\mathbf{V}^{(c)}, \\ \mathbf{W}^{(s)} &= \mathbf{I}^{(s)} + \mathbf{U}^{(s)}\mathbf{V}^{(s)}, \end{aligned}$$

where $\mathbf{I}^{(c)}$ and $\mathbf{I}^{(s)}$ are c- and n-dimensional identity matrices, respectively. The matrices $\mathbf{U}^c$, $\mathbf{V}^c$, $\mathbf{U}^s$, and $\mathbf{V}^s$ are of size $c \times d_c$, $d_c \times c$, $n \times d_s$, and $d_c \times n$, respectively, where d_c and d_s are constant latent dimensions of these four matrices. Therefore, we can rewrite Equation 3.4 as

$$\mathbf{x}_c = (\mathbf{I}^{(c)} + \mathbf{U}^{(c)}\mathbf{V}^{(c)})\mathbf{x}, \tag{3.5}$$

$$\mathbf{z} = \mathbf{x}_c(\mathbf{I}^{(s)} + \mathbf{U}^{(s)}\mathbf{V}^{(s)}). \tag{3.6}$$

We call Eq. 3.6 the Woodbury transformation because the Woodbury matrix identity [111] and Sylvester's determinant identity [102] allow efficient computation of its inverse and Jacobian determinant.

Woodbury matrix identity.[1] Let $\mathbf{I}^{(n)}$ and $\mathbf{I}^{(k)}$ be n- and k-dimensional identity matrices, respectively. Let $\mathbf{U}$ and $\mathbf{V}$ be $n \times k$ and $k \times n$ matrices, respectively. If $\mathbf{I}^{(k)} + \mathbf{VU}$ is invertible, then

$$(\mathbf{I}^{(n)} + \mathbf{UV})^{-1} = \mathbf{I}^{(n)} - \mathbf{U}(\mathbf{I}^{k} + \mathbf{VU})^{-1}\mathbf{V}.$$

Sylvester's determinant identity. Let $\mathbf{I}^{(n)}$ and $\mathbf{I}^{(k)}$ be n- and k-dimensional identity

[1] A more general version replaces $\mathbf{I}^{(n)}$ and $\mathbf{I}^{(k)}$ with arbitrary invertible $n \times n$ and $k \times k$ matrices. But this simplified version is sufficient for our tasks.

matrices, respectively. Let $\mathbf{U}$ and $\mathbf{V}$ be $n \times k$ and $k \times n$ matrices, respectively. Then,

$$\det(\mathbf{I}^{(n)} + \mathbf{UV}) = \det(\mathbf{I}^{(k)} + \mathbf{VU}).$$

Based on these two identities, we can efficiently compute the inverse and Jacobian determinant

$$\begin{aligned} \mathbf{x}_c &= \mathbf{y}(\mathbf{I}^{(s)} - \mathbf{U}^{(s)}(\mathbf{I}^{(d_s)} + \mathbf{V}^{(s)}\mathbf{U}^{(s)})^{-1}\mathbf{V}^{(s)}), \\ \mathbf{x} &= (\mathbf{I}^{(c)} - \mathbf{U}^{(c)}(\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)})^{-1}\mathbf{V}^{(c)})\mathbf{x}_c, \end{aligned} \tag{3.7}$$

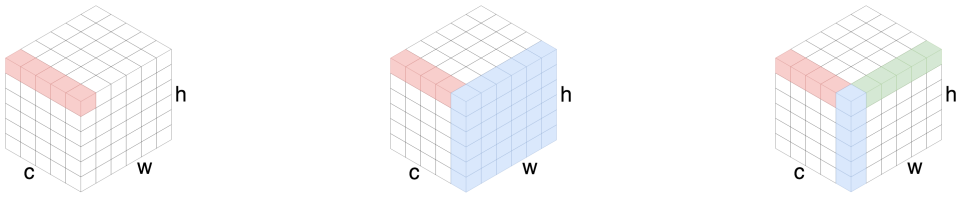
and

$$\log \left| \det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| = n \log \left| \det(\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)}) \right| + c \log \left| \det(\mathbf{I}^{(d_s)} + \mathbf{V}^{(s)}\mathbf{U}^{(s)}) \right|, \tag{3.8}$$

where $\mathbf{I}^{(d_c)}$ and $\mathbf{I}^{(d_s)}$ are d_c- and d_s-dimensional identity matrices, respectively.

A Woodbury transformation is also a generalized permutation layer. We can directly replace an invertible convolution in Figure 2.1a with a Woodbury transformation. In contrast with 1×1 convolutions, Woodbury transformations are able to model correlations along both channel and spatial axes. We illustrate this in Figure 3.1. To implement Woodbury transformations, we need to store four weight matrices, i.e., $\mathbf{U}^{(c)}$, $\mathbf{U}^{(s)}$, $\mathbf{V}^{(c)}$, and $\mathbf{V}^{(s)}$. To simplify our analysis, let $d_c \leq d$ and $d_s \leq d$, where d is a constant. This setting is also consistent with our experiments. The size of $\mathbf{U}^{(c)}$ and $\mathbf{V}^{(c)}$ is $\mathcal{O}(dc)$, and the size of $\mathbf{U}^{(c)}$ and $\mathbf{V}^{(c)}$ is $\mathcal{O}(dn)$. The space complexity is $\mathcal{O}(d(c+n))$.

For training and likelihood computation, the main computational bottleneck is computing $\mathbf{y}$ and the Jacobian determinant. To compute $\mathbf{y}$ with Equation 3.4, we need to first compute the channel transformation and then compute the spatial transformation. The computa-



(a) 1×1 convolution (b) Woodbury (c) ME-Woodbury

Figure 3.1: Visualization of three transformations. The 1×1 convolution only operates along the channel axis. The Woodbury transformation operates along both the channel and spatial axes, modeling the dependencies of one channel directly via one transformation. The ME-Woodbury transformation operates along three axes. It uses two transformations to model spatial dependencies.

tional complexity is $\mathcal{O}(dcn)$. To compute the determinant with Equation 3.8, we need to first compute the matrix product of $\mathbf{V}$ and $\mathbf{U}$, and then compute the determinant. The computational complexity is $\mathcal{O}(d^2(c+n)+d^3)$.

For sampling, we need to compute the inverse transformations, i.e., Equation 3.7. With the Woodbury identity, we actually only need to compute the inverses of $\mathbf{I}^{(d_s)}+\mathbf{V}^{(s)}\mathbf{U}^{(s)}$ and $\mathbf{I}^{(d_c)}+\mathbf{V}^{(c)}\mathbf{U}^{(c)}$, which are computed with time complexity $\mathcal{O}(d^3)$. To implement the inverse transformations, we can compute the matrix chain multiplication, so we can avoid computing the product of two large matrices twice, yielding cost $\mathcal{O}(c^2+n^2)$. For example, for the inverse spatial transformation, we can compute it as $\mathbf{x}_c=\mathbf{z}-((\mathbf{z}\mathbf{U}^{(s)})(\mathbf{I}^{(d_s)}+\mathbf{V}^{(s)}\mathbf{U}^{(s)})^{-1})\mathbf{V}^{(s)}$, so that its complexity is $\mathcal{O}(d^3+cd^2+cnd)$. The total computational complexity of Equation 3.7 is $\mathcal{O}(dcn+d^2(n+c)+d^3)$.

In practice, we found that for a high-dimensional input, a relatively small d is enough to obtain good performance, e.g., the input is $256\times256\times3$ images, and $d=16$. In this situation, $nc\geq d^3$. Therefore, we can omit d and approximately see the spatial complexity as $\mathcal{O}(c+n)$, and the forward or inverse transformation as $\mathcal{O}(nc)$. They are all linear to the input size.

We do not restrict $\mathbf{U}$ and $\mathbf{V}$ to force $\mathbf{W}$ to be invertible. Based on analysis by Hoogeboom

et al. [39], the training maximizes the log-likelihood, which implicitly pushes $\det(\mathbf{I}+\mathbf{VU})$ away from 0. Therefore, it is not necessary to explicitly force invertibility. In our experiments, the Woodbury transformations are as robust as other invertible convolution layers.

3.1.3 Memory-Efficient Variant

In Eq. 3.4, one potential challenge arises from the sizes of $\mathbf{U}^{(s)}$ and $\mathbf{V}^{((s))}$, which are linear in n. The challenge is that n may be large in some practical problems, e.g., high-resolution images. We develop a memory-efficient variant of Woodbury transformations, i.e., ME-Woodbury, to solve this problem.

The difference between ME-Woodbury transformations and Woodbury transformations is that the ME form cannot directly model spatial correlations. As shown in Figure 3.1c, it uses two transformations, for height and width, together to model the spatial correlations. Therefore, for a specific channel k, when two dimensions $\mathbf{x}_{k,i,j}$ and $\mathbf{x}_{k,u,v}$ are in two different heights, and widths, their interaction will be modeled indirectly. In our experiments, we found that this limitation only slightly impacts ME-Woodbury's performance. The transformations are:

$$\begin{aligned}
\mathbf{x}_c &= (\mathbf{I}^{(c)} + \mathbf{U}^{(c)}\mathbf{V}^{(c)})\mathbf{x}, \\
\mathbf{x}_w &= \text{reshape}(\mathbf{x}_c, (ch, w)), \\
\mathbf{x}_w &= \mathbf{x}_c(\mathbf{I}^{(w)} + \mathbf{U}^{(w)}\mathbf{V}^{(w)}), \\
\mathbf{x}_h &= \text{reshape}(\mathbf{x}_w, (cw, h)), \\
\mathbf{z} &= \mathbf{x}_h(\mathbf{I}^{(h)} + \mathbf{U}^{(h)}\mathbf{V}^{(h)}), \\
\mathbf{z} &= \text{reshape}(\mathbf{z}, (c, hw)),
\end{aligned} \tag{3.9}$$

where reshape$(\mathbf{x}, (n, m))$ reshapes $\mathbf{x}$ to be an $n \times m$ matrix. Matrices $\mathbf{I}^{(w)}$ and $\mathbf{I}^{(h)}$ are w- and h-dimensional identity matrices, respectively. Matrices $\mathbf{U}^{(w)}, \mathbf{V}^{(w)}, \mathbf{U}^{(h)}$, and $\mathbf{V}^{(h)}$ are $w \times d_w$, $d_w \times w$, $w \times d_w$, and $d_w \times w$ matrices, respectively, where d_w and d_h are constant latent dimensions.

Using the Woodbury matrix identity and the Sylvester's determinant identity, we can compute the inverse and Jacobian determinant:

$$\begin{aligned}
\mathbf{z} &= \text{reshape}(\mathbf{z}, (cw, h)), \\
\mathbf{x}_h &= \mathbf{z}(\mathbf{I}^{(h)} - \mathbf{U}^{(h)}(\mathbf{I}^{(d_h)} + \mathbf{V}^{(h)}\mathbf{U}^{(h)})^{-1}\mathbf{V}^{(h)}), \\
\mathbf{x}_w &= \text{reshape}(\mathbf{x}_h, (ch, w)), \\
\mathbf{x}_w &= \mathbf{x}_w(\mathbf{I}^{(w)} - \mathbf{U}^{(w)}(\mathbf{I}^{(d_w)} + \mathbf{V}^{(w)}\mathbf{U}^{(w)})^{-1}\mathbf{V}^{(w)}), \\
\mathbf{x}_c &= \text{reshape}(\mathbf{x}_w, (c, hw)), \\
\mathbf{x} &= (\mathbf{I}^{(c)} - \mathbf{U}^{(c)}(\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)})^{-1}\mathbf{V}^{(c)})\mathbf{x}_c,
\end{aligned} \tag{3.10}$$

$$\begin{aligned}
\log \left| \det\left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right) \right| &= hw \log \left| \det(\mathbf{I}^{(d_c)} + \mathbf{V}^{(c)}\mathbf{U}^{(c)}) \right| + ch \log \left| \det(\mathbf{I}^{(d_w)} + \mathbf{V}^{(w)}\mathbf{U}^{(w)}) \right| \\
&\quad + cw \log \left| \det\left(\mathbf{I}^{(d_h)} + \mathbf{V}^{(h)}\mathbf{U}^{(h)}\right) \right|,
\end{aligned} \tag{3.11}$$

where $\mathbf{I}^{(d_w)}$ and $\mathbf{I}^{(d_h)}$ are d_w- and d_h-dimensional identity matrices, respectively. The Jacobian of the reshape() is an identity matrix, so its log-determinant is 0.

We call Equation 3.9 the memory-efficient Woodbury transformation because it reduces space complexity from $\mathcal{O}(c + hw)$ to $\mathcal{O}(c + h + w)$. This method is effective when h and w are large. To analyze its complexity, we let all latent dimensions be less than d as before. The complexity of forward transformation is $\mathcal{O}(dchw)$; the complexity of computing the determinant is $\mathcal{O}(d(c+h+w)+d^3)$; and the complexity of computing the inverse is $\mathcal{O}(dchw+$

$d^2(c + ch + cw) + d^3)$. The same as Woodbury transformations, when the input is high dimensional, we can omit d. Therefore, the computational complexities of the memory-efficient Woodbury transformation are also linear with the input size.

3.2 Related Work

Rezende and Mohamed [92] developed planar flows for variational inference $\mathbf{z}_{t+1} = \mathbf{z}_t + \mathbf{u}\delta(\mathbf{w}^T\mathbf{z}_t + b)$, where $\mathbf{z}$, $\mathbf{w}$, and $\mathbf{u}$ are d-dimensional vectors, $\delta()$ is an activation function, and b is a scalar.

Berg et al. [12] generalized these to Sylvester flows $\mathbf{z}_{t+1} = \mathbf{z}_t + \mathbf{Q}\mathbf{R}\delta(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{z}_t + \mathbf{r})$, where $\mathbf{R}$ and $\tilde{\mathbf{R}}$ are upper triangular matrices, $\mathbf{Q}$ is composed of a set of orthonormal vectors, and $\mathbf{r}$ is a d-dimensional vector. The resulting Jacobian determinant can be efficiently computed via Sylvester's identity, just as our methods do. However, Woodbury transformations have key differences from Sylvester flows. First, Berg et al. only analyze their models on vectors. The inputs to our layers are matrices, so our method operates on high-dimensional input, e.g., images. Second, though Sylvester flows are inverse functions, computing their inverse is difficult. One possible way is to apply iterative methods [9, 17, 101] to compute the inverse. But this research direction is unexplored. Our layers can be inverted efficiently with the Woodbury identity. Third, our layers do not restrict the transformation matrices to be triangular or orthogonal. In fact, Woodbury transformations can be seen as another generalized variant of planar flows on matrices, with $\delta(\mathbf{x}) = \mathbf{x}$, and whose inverse is tractable. Roughly speaking, Woodbury transformations can also be viewed as applying the planar flows sequentially to each row of the input matrix. After this work was completed and submitted, we learned that the TensorFlow software [1] also uses the Woodbury identity in their affine bijector.

Aside from low rank factorization, i.e., Equation 3.5, there may have other ways to factorize the weight matrices. For example, Oliva et al. use LU decomposition. When using LU decomposition, the complexity of computing Jacobian determinant is $\mathcal{O}(n)$, but the complexity of computing forward transformation, i.e., $\mathbf{z} = \mathbf{x}\mathbf{W}$, is $\mathcal{O}(n^2)$, since the decomposed triangular matrices are still $n \times n$ in size. Therefore, in theory, LU decomposition should be slightly slower than Woodbury transformations.

Normalizing flows have also been used for variational inference, density estimation, and generative modeling. Autoregressive flows [41, 52, 69, 85] restrict each variable to depend on those that precede it in a sequence, forcing a triangular Jacobian. Non-linear coupling layers replace the affine transformation function. Specifically, spline flows [25, 78] use spline interpolation, and Flow++ [37] uses a mixture cumulative distribution function to define these functions. Flow++ also uses variational dequantization to prevent model collapse. Many works [27, 39, 47, 51] develop invertible convolutional flows to model interactions among dimensions. MintNet [101] is a flexible architecture composed of multiple masked invertible layers. I-ResNet [9, 17] uses discriminative deep network architecture as the flow. These two models require iterative methods to compute the inverse. Discrete flows [40, 104] and latent flows [130] can be applied to discrete data such as text. Continuous-time flows [16, 33] have been developed based on the theory of ordinary differential equations.

3.3 Experiments

In this section, we compare the performance of Woodbury transformations against other modern flow architectures, measuring running time, bit per-dimension ($\log_2$-likelihood), and sample quality.

Running Time We follow Finz et al. [27] and compare the per-sample running time of

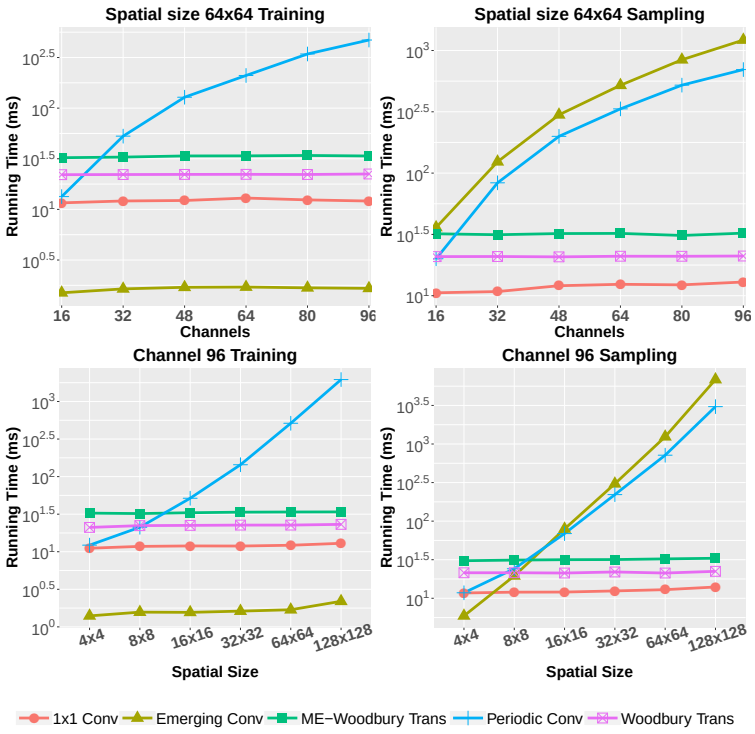


Figure 3.2: Running time comparison. Sampling with emerging convolutions is slow, since their inverses are not parallelizable. Periodic convolutions are costly for larger inputs. Both 1×1 convolutions and Woodbury transformations are efficient in training and sampling.

Woodbury transformations to other generalized permutations: 1×1 [51], emerging [39], and periodic convolutions [27, 39]. We test the training time and sampling time. In training, we compute (1) forward propagation, i.e., $\mathbf{z} = \mathbf{f}(\mathbf{x})$, of a given function $\mathbf{f}()$, (2) the Jacobian determinant, i.e., $\det\left(\left|\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right|\right)$, and (3) the gradient of parameters. For sampling, we compute the inverse of transformation $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{z})$. For emerging and periodic convolutions, we use 3×3 kernels. For Woodbury transformations, we fix the latent dimension $d = 16$. For fair comparison, we implement all methods in Pytorch and run them on an Nvidia Titan V GPU. We follow Hoogeboom et al. [39] and implement the emerging convolution inverse in Cython, and we compute it on a 4 Ghz CPU (the GPU version is slower than the Cython version). We first fix the spatial size to be 64×64 and vary the channel number. We then fix the channel number to be 96 and vary the spatial size.

The results are shown in Figure 3.2. For training, the emerging convolution is the fastest. This is because its Jacobian is a triangular matrix, so computing its determinant is much more efficient than other methods. The Woodbury transformation and ME-Woodbury are slightly slower than the 1x1 convolution, since they contain more transformations. Emerging convolutions, Woodbury transformations, and 1x1 convolutions only slightly increase with input size, rather than increasing with $\mathcal{O}(c^3)$. This invariance to input size is likely because of how the GPU parallelizes computation. The periodic convolution is efficient only when the input size is small. When the size is large, it becomes slow, e.g., when the input size is $96 \times 64 \times 64$, it is around 30 times slower than Woodbury transformations. In our experiments, we found that the Fourier transformation requires a large amount of memory. According to Finz et al. [27], the Fourier step may be the bottleneck that impacts periodic convolution's scalability. A more efficient implementation of Fourier transformation, e.g., [47], may improve its running time.

For sampling, both 1×1 convolutions and Woodbury transformations are efficient. The 1×1 convolution is the fastest, and the Woodbury transformations are only slightly slower. Neither is sensitive to the change of input size. Emerging convolutions and periodic convolutions are much slower than Woodbury transformations, and their running time increases with the input size. When the input size is $96 \times 128 \times 128$, they are around 100 to 200 times slower than Woodbury transformations. This difference is because emerging convolutions cannot make use of parallelization, and periodic transformations require conversion to Fourier form. Based on these results, we can conclude that both emerging convolution and periodic convolution do not scale well to high-dimensional inputs. In contrast, Woodbury transformations are efficient in both training and sampling.

Quantitative Evaluation We compare Woodbury transformations with state-of-the-art flow models, measuring bit per-dimension (bpd). We train with the CIFAR-10 [57] and

ImageNet [95] datasets. We compare with three generalized permutation methods—1×1 convolution, emerging convolution, and periodic convolution—and two coupling layers—neural spline coupling [25] and MaCow [69]. We use Glow (Fig. 2.1, [51]) as the basic flow architecture. For each method, we replace the corresponding layer. For example, to construct a flow with Woodbury transformations, we replace the 1×1 convolution with a Woodbury transformation, i.e., Eq. 3.4. For all generalized permutation methods, we use affine coupling. For each of the coupling layer baselines, we substitute it for the affine coupling. We tune the parameters of neural spline coupling and MaCow so that their sizes are close to affine coupling. We follow Hoogeboom et al. [39] and test the performance of small models. For 32×32 images, we set the number of levels to $L = 3$ and the number of steps per-level to $K = 8$. For 64×64 images, we use $L = 4$ and $K = 16$. More details are in the appendix.

Table 3.1: Quantitative evaluation results.

	Quantitative measure (bpd)			**Model sizes (# parameters)**	
	CIFAR-10 32x32	ImageNet 32x32	ImageNet 64x64	32x32 images	64x64 images
1×1 convolution	3.51	4.32	3.94	11.02M	37.04M
Emerging	3.48	4.26	3.91	11.43M	40.37M
Periodic	3.49	4.28	3.92	11.21M	38.61M
Neural spline	3.50	4.24	3.95	**10.91M**	38.31M
MaCow	3.48	4.34	4.15	11.43M	37.83M
ME-Woodbury	3.48	4.22	3.91	11.02M	**36.98M**
Woodbury	**3.47**	**4.20**	**3.87**	11.10M	37.60M

The test-set likelihoods are listed in Table 3.1 left. Our scores are worse than those reported by Hoogeboom et al. [39], Kingma and Dhariwal [51] because we use smaller models and train each model on a single GPU. Based on the scores, 1×1 convolutions perform the worst. Emerging convolutions and periodic convolutions score better than the 1×1 convolutions, since they are more flexible and can model the dependencies along the spatial axes. Neural

spline coupling works well on 32×32 images, but do slightly worse than 1×1 convolution on 64×64 images. MaCow does not work well on ImageNet. This trend demonstrates the importance of permutation layers. They can model the interactions among dimensions and shuffle them, which coupling layers cannot do. Without a good permutation layer, a better coupling layer still cannot always improve the performance. The Woodbury transformation models perform the best, likely because they can model the interactions between the target dimension and all other dimensions, while the invertible convolutions only model the interactions between target dimension its neighbors. ME-Woodbury performs only slightly worse than the full version, showing that its restrictions provide a useful tradeoff between model quality and efficiency.

We list model sizes in Table 3.1 (right). Despite modeling rich interactions, Woodbury transformations are not the largest. With 32×32 images, ME-Woodbury and 1×1 convolution are the same size. When the image size is 64×64, ME-Woodbury is the smallest. This is because we use the multi-scale architecture, i.e., Fig. 2.1, to combine layers. The squeeze layer doubles the input variable's channels at each level, so larger L suggests larger c. The space complexities of invertible convolutions are $\mathcal{O}(c^2)$, while the space complexity of ME-Woodbury is linear to c. When c is large, the weight matrices of invertible convolutions are larger than the weight matrices of ME-Woodbury.

Table 3.2: Evaluation of different d (bits per-dimension).

	Woodbury	ME-Woodbury
$d = 2$	3.54	3.53
$d = 4$	3.51	3.51
$d = 8$	3.48	3.48
$d = 16$	3.47	3.48
$d = 32$	3.47	3.48

Latent Dimension Evaluation We test the impact of latent dimension d on the perfor-

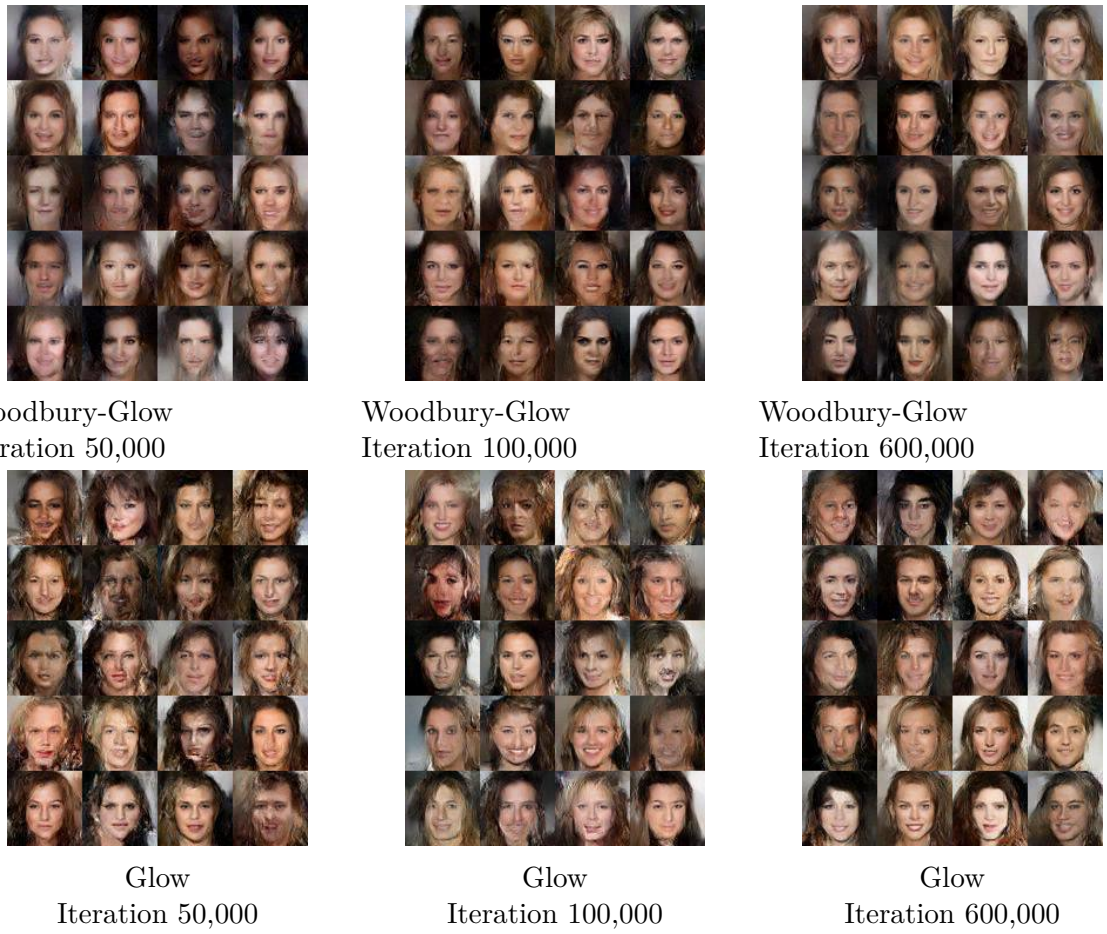


Figure 3.3: Random samples 64×64 drawn from models trained on CelebA with temperature 0.7.

mance of Woodbury-Glow. We train our models on CIFAR-10, and use bpd as metric. We vary d within $\{2, 4, 8, 16, 32\}$. The results are in Table 3.2. When $d < 8$, the model performance will be impacted. When $d > 16$, increasing d will not improve the bpd. This is probably because when d is too small, the latent features cannot represent the input variables well, and when d is too big, the models become hard to train. When $8 \leq d \leq 16$, the Woodbury transformations are powerful enough to model the interactions among dimensions. We also test two values of d, i.e., $16, 32$, of Woodbury-Glow on ImageNet 64×64. The bpds of both d are 3.87, which are consistent with our conclusion.

Sample Quality Comparisons We train Glow and Woodbury-Glow on the CelebA-HQ

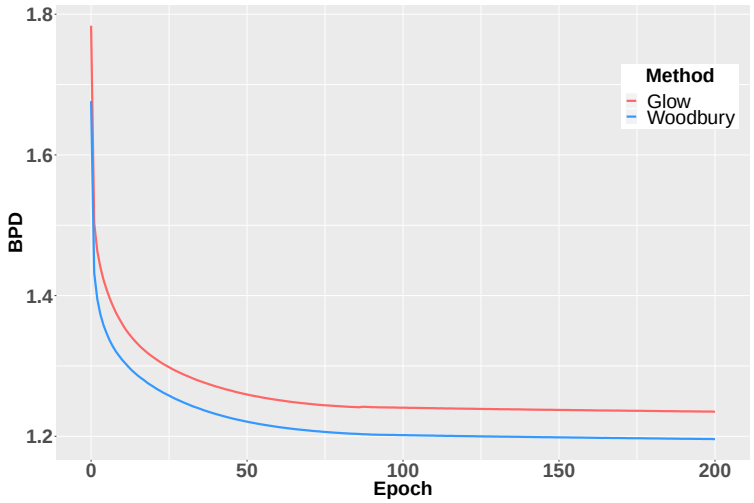


Figure 3.4: Learning curves on CelebA-HQ 64x64. The NLL of Woodbury Glow decreases faster than Glow.

dataset [48]. We use 5-bit images and set the size of images to be 64×64, 128×128, and 256×256. Due to our limited computing resources, we use relatively small models in our experiments. We follow Kingma and Dhariwal [51] and choose a temperature parameter to encourage higher quality samples. Detailed parameter settings are in the appendix. We compare samples from Glow and Woodbury-Glow during three phases of training, displayed in Fig. 3.3. The samples show a clear trend where Woodbury-Glow more quickly learns to generate reasonable face shapes. After 100,000 iterations, it can already generate reasonable samples, while Glow's samples are heavily distorted. Woodbury-Glow samples are consistently smoother and more realistic than samples from Glow in all phases of training. The samples demonstrate Woodbury transformations' advantages. The learning curves in Figure 3.4 also show that the NLL of Woodbury Glow decreases faster, which is consistent to the sample comparisons. In the appendix, we show analogous comparisons using higher resolution versions of CelebA data, which also exhibit the trend of Woodbury-Glow generating more realistic images than Glow at the same training iterations.

3.4 Conclusion

In this chapter, we introduce Woodbury transformations, which use the Woodbury matrix identity to compute the inverse transformations and Sylvester's determinant identity to compute Jacobian determinants. Our method has the same advantages as invertible $d \times d$ convolutions that can capture correlations among all dimensions. In contrast to the invertible $d \times d$ convolutions, our method is parallelizable and the computational complexity of our methods are linear to the input size, so that it is still efficient in computation when the input is high-dimensional. One potential limitation is that Woodbury transformations do not have parameter sharing scheme as in convolutional layers, so one potential future research is to develop partially Woodbury transformations that can share parameters. We test our models on multiple image datasets and they outperform state-of-the-art methods.

Chapter 4

Structured Output Learning with Conditional Generative Flows

Structured output learning, i.e., structured prediction, is an advanced supervised learning problem, whose goal is to model a mapping from the input $\mathbf{x}$ to the high-dimensional structured output $\mathbf{y}$. Many computer vision or natural language processing tasks such as image segmentation [81] and sequence labeling [60] can be interpreted as structured prediction problems. In many such problems, it is also important to make diverse predictions to capture the variability of plausible solutions to the structured output problem [100].

Many existing methods for structured output learning use graphical models, such as conditional random fields (CRFs) [108], and approximate the conditional distribution $p(y|x)$. Approximation is necessary because, for most graphical models, computing the exact likelihood is intractable. Recently, deep structured prediction models [10, 15, 32, 100, 109, 126] combine deep neural networks with graphical models, using the power of deep neural networks to extract high-quality features and graphical models to model correlations and dependencies among variables. The main drawback of these approaches is that, due to the intractable likelihood, they are difficult to train. Training them requires the construction of surrogate objectives, or approximating the likelihood by using variational inference to infer latent variables. Moreover, once the model is trained, inference and sampling from CRFs require expensive iterative procedures [55].

In this chapter, we develop conditional generative flows (c-Glow) for structured output learning. Our model is a variant of Glow [51], with additional neural networks for capturing the relationship between input features and structured output variables. Compared to most methods for structured output learning, c-Glow has the unique advantage that it can directly model the conditional distribution $p(y|x)$ without restrictive assumptions (e.g., variables being fully connected [56]). We can train c-Glow by exploiting the fact that invertible flows allow exact computation of log-likelihood, removing the need for surrogates or inference. Compared to other methods using normalizing flows (e.g., [51, 105]), c-Glow's output label y is conditioned on both complex input and a high-dimensional tensor rather than a one-dimensional scalar. We evaluate c-Glow on five structured prediction tasks: binary segmentation, multi-class segmentation, color image denoising, depth refinement, and image inpainting, finding that c-Glow's exact likelihood training is able to learn models that efficiently predict structured outputs of comparable quality to state-of-the-art deep structured prediction approaches.

4.1 Conditional Generative Flows for Structured Output Learning

In Chapter 2 we introduce Glow [51], i.e., Figure 2.1, which is a popular flow based generative model for density estimation. In this section, we introduce our conditional generative flow (c-Glow), which is a variant of Glow for structured prediction.

4.1.1 Conditional Glow

To modify Glow to be a conditional generative flow, i.e., Figure 2.5, we need to add conditioning architectures to its three components: the actnorm layer, the 1×1 convolutional layer, and the affine coupling layer. The main idea is to use a neural network, which we refer to as a *conditioning network* (CN), to generate the parameter weights for each layer, conditioned on the input variable $\mathbf{x}$. The details are as follows.

Conditional actnorm. The parameters of an actnorm layer are two $1 \times c$ vectors, i.e., the scale $\mathbf{s}$ and the bias $\mathbf{b}$. In conditional Glow, we use a CN to generate these two vectors and then use them to transform the variable, i.e.,

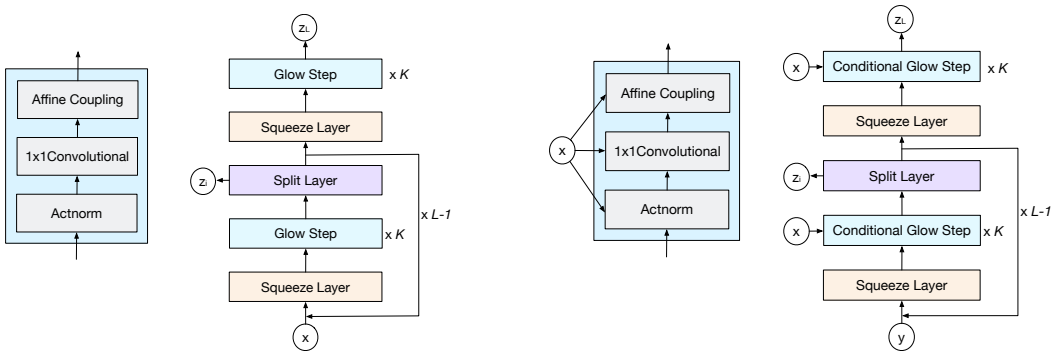
$$\begin{aligned} \mathbf{s}, \mathbf{b} &= \text{CN}(\mathbf{x}), \\ \mathbf{z}_{i,j} &= \mathbf{s} \odot \mathbf{y}_{i,j} + \mathbf{b}. \end{aligned}$$

where $\mathbf{z}$ and $\mathbf{y}$ are the latent code and input of a conditional flow layer, and the $\mathbf{x}$ is the input of CN.

Conditional 1×1 convolutional. The 1×1 convolutional layer uses a $c \times c$ weight matrix to permute each spatial dimension's variable. In conditional Glow, we use a conditioning network to generate this matrix:

$$\begin{aligned} \mathbf{W} &= \text{CN}(\mathbf{x}), \\ \mathbf{z}_{i,j} &= \mathbf{W}\mathbf{y}_{i,j}. \end{aligned}$$

Conditional affine coupling. The affine coupling layer separates the input variable into two halves, i.e., v_1 and v_2. It uses v_1 as the input to an NN to generate scale and bias parameters for v_2. To build a conditional affine coupling layer, we use a CN to extract



(a) Glow architecture (b) c-Glow architecture

Figure 4.1: Model architectures for Glow and conditional Glow. For each model, the left sub-graph is the architecture of each step, and the right sub-graph is the whole architecture. The parameter L represents the number of levels, and K represents the depth of each level.

features from x, and then we concatenate it with v_1 to form the input of NN.

$$\begin{aligned}
\mathbf{y}_1, \mathbf{y}_2 &= \text{split}(\mathbf{y}), \\
\mathbf{x}_r &= \text{CN}(\mathbf{x}), \\
\mathbf{s}_2, \mathbf{b}_2 &= \text{NN}(\mathbf{y}_1, \mathbf{x}_r), \\
\mathbf{z}_2 &= \mathbf{s}_2 \odot \mathbf{y}_2 + \mathbf{b}_2, \\
\mathbf{z} &= \text{concat}(\mathbf{z}_1, \mathbf{y}_2).
\end{aligned}$$

We can still use the multi-scale architecture to combine these conditional components to preserve the efficiency of computation. Figure 4.1 illustrates the Glow and c-Glow architectures for comparison.

Since the conditioning networks do not need to be invertible when optimizing a conditional model, we define the general approach without restrictions to their architectures here. Any differentiable network suffices and preserves the ability of c-Glow to compute the exact conditional likelihood of each input-output pair. We will specify the architectures we use in our experiments in Section 4.3.1.

4.1.2 Learning

To learn the model parameters, we can take advantage of the efficiently computable log-likelihood for flow-based models. In cases where the output is continuous, the likelihood calculation is direct. Therefore, we can back-propagate to differentiate the exact conditional likelihood, i.e., Eq. 2.5, and optimize all c-Glow parameters using gradient methods.

In cases where the output is discrete, we follow [22, 37, 51] and add uniform noise to y during training to dequantize the data. This procedure augments the dataset and prevents model collapse. We can still use back-propagation and gradient methods to optimize the likelihood of this approximate continuous distribution. By expanding the proofs by Theis et al. (2015) and Ho et al. (2019), we can show that the discrete distribution is lower-bounded by this continuous distribution.

With a slight abuse of notation, we let $q(\mathbf{y}|\mathbf{x})$ be our discrete hypothesis distribution and $p(\mathbf{v}|\mathbf{x})$ be the dequantized continuous model. Then our goal is to maximize the likelihood q, which can be expressed by marginalizing over values of $\mathbf{v}$ that round to $\mathbf{y}$:

$$q(\mathbf{y}|\mathbf{x}) = \int_{\mathbf{u}\in[-0.5,0.5)^d} p(\mathbf{y}+\mathbf{u}|x)d\mathbf{u},$$

where d is the variable's dimension, and $\mathbf{u}$ represents the difference between the continuous variable $\mathbf{v}$ and the rounded, quantized $\mathbf{y}$.

Let $p_d(\mathbf{x}, \mathbf{y})$ be the true data distribution, and $\tilde{p}_d(\mathbf{x}, \mathbf{y})$ be the distribution of the dequantized dataset. The learning process maximizes $\mathbb{E}_{\tilde{p}_d(\mathbf{x},\mathbf{y})}[\log p(\mathbf{v}|\mathbf{x})]$. We expand this and apply

Jensen's Inequality to obtain the bound:

$$\begin{aligned} & \mathbb{E}_{\tilde{p}_d(\mathbf{x},\mathbf{y})}[\log p(\mathbf{v}|\mathbf{x})] \\ = & \int_{\mathbf{x}} \sum_{\mathbf{y}} p_d(\mathbf{x},\mathbf{y}) d\mathbf{x} \int_{\mathbf{u}} \log p(\mathbf{y}+\mathbf{u}|x) d\mathbf{u} \\ \leq & \int_{\mathbf{x}} \sum_{\mathbf{y}} p_d(\mathbf{x},\mathbf{y}) d\mathbf{x} \log \int_{\mathbf{u}} p(\mathbf{y}+\mathbf{u}|\mathbf{x}) d\mathbf{u} \\ = & \mathbb{E}_{p_d(\mathbf{x},\mathbf{y})}[\log q(\mathbf{y}|\mathbf{x})]. \end{aligned}$$

Therefore, when y is discrete, the learning optimization, which maximizes the continuous likelihood $p(v|x)$, maximizes a lower bound on $q(y|x)$.

4.1.3 Inference

Given a learned model $p(\mathbf{y}|\mathbf{x})$, we can perform efficient sampling with a single forward pass through the c-Glow. We first calculate the transformation functions given $\mathbf{x}$ and then sample the latent code $\mathbf{z}$ from $p_Z(\mathbf{z})$. Finally, we propagate the sampled $\mathbf{z}$ through the model, and we get the corresponding sample $\mathbf{y}$. The whole process can be summarized as

$$\mathbf{z} \sim p_Z(\mathbf{z}), \mathbf{y} = g_{\mathbf{x},\phi}(\mathbf{z}), \tag{4.1}$$

where $g_{\mathbf{x},\phi} = f^{-1}_{\mathbf{x},\phi}$ is the inverse function.

The core task in structured output learning is to predict the best output, i.e., $\mathbf{y}^*$, for an input $\mathbf{x}$. This process can be formalized as looking for an optimized $\mathbf{y}^*$ such that

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}). \tag{4.2}$$

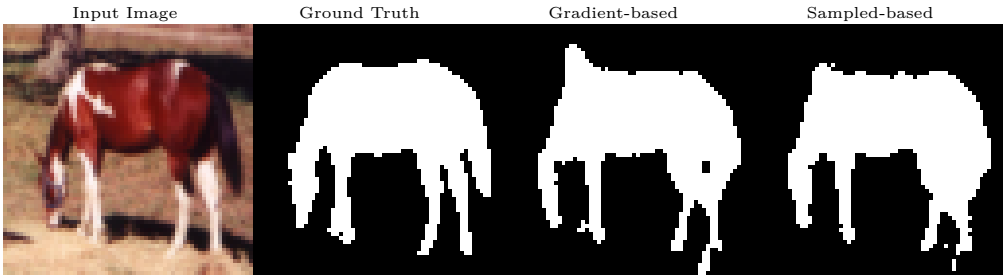


Figure 4.2: Illustration of difference between a gradient-based method and a sample-based method. From left to right: the input image, the ground truth label, the gradient-based prediction, and the sample-based prediction. In the third image, the horse has a horn on its back. This is because the gradient-based method is trapped into a local optimum, which assumes the head of this horse should be in that place. In the fourth image, the sample average smooths out the horn because most samples do not have the horn mistake.

To compute Equation 4.2, we can use gradient-based optimization, e.g., to optimize y based on gradient descent. However, in our experiments, we found that this method is always slow, i.e., it takes thousands of iterations to converge. Worse, since the probability density function is non-convex with a highly multi-modal surface, it often gets stuck in local optima, resulting in sub-optimal prediction. Therefore, we use a sample-based method to approximate the inference instead. Let $\{z_1, ..., z_M\}$ be samples drawn from $p_Z(z)$. Estimated marginal expectations for each variable can be computed from the average:

$$\mathbf{y}^* \approx \frac{1}{M} \sum_{i=1}^{M} g_{\mathbf{x},\phi}(\mathbf{z}_i). \tag{4.3}$$

This sample-based method can overcome the gradient-based method's problems. In our experiments, we found that we only need 10 samples to get a high quality prediction, so inference is faster. The sample average can smooth out some anomalous values, further improving prediction. One illustration of difference between the gradient-based method and the sample-based method is in Figure 4.2.

When $\mathbf{y}$ is a continuous variable, we can directly get $\mathbf{y}^*$ from the above sample-based prediction. When $\mathbf{y}$ is discrete, we follow previous literature [10, 34] to round $\mathbf{y}^*$ to discrete

values. In our experiments, we find that the predicted $\mathbf{y}^*$ values are already near integral values.

4.2 Related Work

In this section, we introduce research that most related to c-Glow.

4.2.1 Deep Structured Models

One emerging strategy to construct deep structured models is to combine deep neural networks with graphical models. However, this kind of model can be difficult to train, since the likelihood of graphical models is usually intractable. Chen et al. (2015) proposed joint learning approaches that blend the learning and approximate inference to alleviate some of these computational challenges. Zheng et al. (2015) proposed CRF-RNN, a method that treats mean-field variational CRF inference as a recurrent neural network to allow gradient-based learning of model parameters. Wang et al. (2016) proposed proximal methods for inference. And Sohn et al. (2015) used variational autoencoders [50] to generate latent variables for predicting the output. While using a surrogate for the true likelihood is generally viewed as a concession, Norouzi et al. (2016) found that training with a tractable task-specific loss often yielded better performance for the goal of reducing specific task losses than training with general-purpose likelihood approximations. Their analysis hints that fitting a distribution with a true likelihood may not always train the best predictor for specific tasks.

Another direction combining structured output learning with deep models is to construct energy functions with deep networks. Structured prediction energy networks (SPENs) [10] define energy functions for scoring structured outputs as differentiable deep networks. The

likelihood of a SPEN is intractable, so the authors used structured SVM loss to learn. SPENs can also be trained in an end-to-end learning framework [11] based on unrolled optimization. Methods to alleviate the cost of SPEN inference include replacing the argmax inference with an inference network [107]. Inspired by Q-learning, Gygli et al. (2017) used an oracle value function as the objective for energy-based deep networks. Graber et al. (2018) generalized SPENs by adding non-linear transformations on top of the score function.

4.2.2 Conditional Normalizing Flows

Trippe and Turner (2018) first develop conditional normalizing flows and use it to solve the one-dimensional regression problem. Our method is different from theirs in that the labels in our problem are high-dimensional tensors rather than scalars. Recently, the idea of applying conditional flows to structured prediction is developed concurrently and independently by [110]. The main difference between their work and c-Glow is they only use one type of conditional flow layers, but we use three types of conditional flow layers.

C-Glow is a pioneer work in this area that uses conditional normalizing flows to model high-dimensional structured data. The idea of using conditioning networks is similar to HyperNetworks [35]. The difference is c-Glow generates weights for flow layers. After c-Glow, many other conditional flow models are developed for a variety of problems. Li et al. (2019) develop flow-based models for arbitrary conditional likelihoods. FlowSeq [70] applies conditional flows to non-autoregressive sequence-to-sequence generation. C-Flow [86] contains two flow branches, and can generate data from two domains. It can be applied to conditional generation tasks such as image to 3D point clouds generation. Srflow [68] is specifically designed for image super-resolution. Zand et al. develop MotionFlow for an autoregressive structred prediction task – human motion prediction.

Aside from the flow layers introduced in Section 4.1.1, other unconditional flow layers may also be modified to conditional layers. For example, to develop conditional Woodbury layers, we can use conditioning networks to generate the $\mathbf{U}$ and $\mathbf{V}$ weight matrices.

4.3 Experiments

In this section, we evaluate c-Glow on five structured prediction tasks: binary segmentation, multi-class segmentation, image denoising, depth refinement, and image inpainting. We find c-Glow is among the class of state-of-the-art methods while retaining its likelihood and sampling benefits.

4.3.1 Architecture and Setup

To specify a c-Glow architecture, we need to define conditioning networks that generate weights for the conditional actnorm, 1×1 convolutional, and affine layers.

For the conditional actnorm layer, we use a six-layer conditioning network. The first three layers are convolutional layers that downscale the input $\mathbf{x}$ to a reasonable size. The last three layers are then fully connected layers, which transform the resized $\mathbf{x}$ to the scale $\mathbf{s}$ and the bias $\mathbf{b}$ vectors. For the downscaling convolutional layers, we use a simple method to determine their kernel size and stride. Let H_i and H_o be the input and output sizes. Then we set the stride to H_i/H_o and the kernel size to $2 \times \text{padding} + \text{stride}$.

For the conditional 1×1 convolutional layer, we use a similar six-layer network to generate the weight matrix. The only difference is that the last fully connected layer will generate the weight matrix $\mathbf{W}$. For the actnorm and 1×1 convolutional conditional networks, the number of channels of the convolutional layers, i.e., n_c, and the width of the fully connected

layers, i.e., n_w, will impact the model's performance.

For the conditional affine layer, we use a three-layer conditional network to extract features from $\mathbf{x}$, and we concatenate it with $\mathbf{y}_1$. Among the three layers, the first and the last layers use 3×3 kernels. The middle layer is a downscaling convolutional layer. We vary the number of channels of this conditional network to be $\{8, 16, 32\}$, and we find that the model is not very sensitive to this variation. In our experiments, we fix it to have 16 channels. The affine layer itself is composed of three convolutional layers with 256 channels.

We use the same multi-scale architecture as Glow to connect the layers, so the number of levels L and the number of steps of each level K will also impact the model's performance. We use Adam [49] to tune the learning rates, with $\alpha = 0.0002$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We set the mini-batch size to be 2. Based on our empirical results, these settings allow the model to converge quickly. For the experiments on small datasets, i.e., semantic segmentation and image denoising, we run the program for 5×10^4 iterations to guarantee the algorithms have fully converged. For the experiments on inpainting, the training set is large, so we run the program for 3×10^5 iterations.

4.3.2 Binary Segmentation

In this set of experiments, we use the Weizmann Horse Image Database [14], which contains 328 images of horses and their segmentation masks indicating whether pixels are part of horses or not. The training set contains 200 images, and the test set contains 128 images. We compare c-Glow with DVN [34], NLStruct [32], and FCN[1] [66]. Since the code for DVN and NLStruct is not available online, we reproduce results of DVN and NLStruct by Gygli et al. (2017), and Graber et al. (2018). We use mean intersection-over-union (IOU) as the

[1]We use code from https://github.com/wkentaro/pytorch-fcn.

metric. We resize the images and masks to be 32×32, 64×64, and 128×128 pixels. For c-Glow, we follow Kingma and Dhariwal (2018) to preprocess the masks; we copy each mask three times and tile them together, so y has three channels. This transformation can improve the model performance. We set $L = 3$, $K = 8$, $n_c = 64$, and $n_w = 128$.

Table 4.1: Binary segmentation results (IOU).

Image Size	**c-Glow**	**FCN**	**DVN**	**NLStruct**
32×32	0.812	0.558	0.840	—
64×64	0.852	0.701	—	0.752
128×128	0.858	0.795	—	—

Table 4.1 lists the results. DVN only has result on 32×32 images, and NLStruct only has result on 64×64 images. The NLStruct is tested on a smaller test set with 66 images. In our experiments, we found that the smaller test set does not have significant impact on the IOUs. DVN and NLStruct are deep energy-based models. FCN is a feed-forward deep model specifically designed for semantic segmentation. Energy-based models outperform FCN, because they use energy functions to capture the dependencies among output labels. Specifically, DVN performs the best on 32×32 images. The papers on DVN and NLStruct do not include results for large images. Thus, we only include small image results for DVN and NLStruct. C-Glow can easily handle larger size structured prediction tasks, e.g., 128×128 images. Even though c-Glow performs slightly worse than DVN on small images, it significantly outperforms FCN and NLStruct on larger images. The IOUs of c-Glow on larger images are also better than DVN on small images.

4.3.3 Multi-class Segmentation

In this set of experiments, we use the Labeled Faces in the Wild (LFW) dataset [42, 46]. It contains 2,927 images of faces, which are segmented into three classes: face, hair, and background. We use the same training, validation, and test split as previous works [34, 46], and super-pixel accuracy (SPA) as our metric. Since c-Glow predicts the pixel-wise label, we follow previous papers [34, 106] and use the most frequent label in a super-pixel as its class. We resize the images and masks to be 32×32, 64×64, and 128×128 pixels. We compare our method with DVN and FCN. For c-Glow, we set $L = 4$, $K = 8$, $n_c = 64$, and $n_w = 128$. Note that comparing with binary segmentation experiments, we increase the model size by adding one more level. This is because the LFW dataset is larger and multi-class segmentation is more complicated.

Table 4.2: Multi-class segmentation results (SPA).

Image Size	**c-Glow**	**FCN**	**DVN**
32×32	0.914	0.745	0.924
64×64	0.931	0.792	—
128×128	0.945	0.951	—

The results are in Table 4.2. On 32×32 images, DVN performs the best, but c-Glow is comparable. C-glow performs better than FCN on 64×64 images, but slightly worse than FCN on 128×128 images. FCN performs well on large images, but worse than other methods on small images. We attribute this to two reasons. First, for small images, the input features do not contain enough information. The inferences of c-Glow and DVN combine the features as well as the dependencies among output labels to lead to better results. In contrast, FCN predicts each output independently, so it is not able to capture the relationship among output variables. On larger images, the higher resolution makes segmented regions wider in pixels [34, 66], so a feed-forward network that produces coarser and smooth predictions can

perform well. C-Glow's performance is stable. Whether on small images or large images, it is able to generate good quality results. Even though it is slightly worse than the best methods on 32×32 and 128×128 images, it significantly outperforms FCN on 64×64 images. Moreover, c-Glow's SPAs are better than DVN on small images.

4.3.4 Color Image Denoising

In this section, we conduct color image denoising on the BSDS500 dataset [5]. We train models on 400 images and test them on the commonly used 68 images [94]. Following previous work [98], we crop a 256×256 region for each image and resize it to 128×128. We then add Gaussian noise with standard deviation $\sigma = 25$ to each image. We use peak signal-to-noise ratio (PSNR) as our metric, where higher PSNR is better. We compare c-Glow with some state-of-the-art baselines, including BM3D [19], DnCNN [125], and McWNNM [116]. DnCNN is a deep feed-forward model specifically designed for image denoising. BM3D and McWNNM are traditional non-deep models for image denoising. For c-Glow, we set $L = 3$, $K = 8$, $n_c = 64$, and $n_w = 128$. Let $\mathbf{x}$ be the clean images and $\hat{\mathbf{x}}$ be the noisy images. To train the model, we follow Zhang et al. (2017) and use $(\hat{\mathbf{x}})$ as the input and $\hat{\mathbf{x}} - \mathbf{x}$ as the output. To denoise the images, we first predict $\mathbf{y}^*$ and then compute $(\hat{\mathbf{x}} - \mathbf{y}^*)$.

Table 4.3: Color image denoising results (PSNR).

c-Glow	**McWNNM**	**BM3D**	**DnCNN**
27.61	25.58	28.21	28.53

The PSNR comparisons are in Table 4.3. C-Glow produces reasonably good results. However, it is worse than DnCNN and BM3D. To further analyze c-Glow's performance, we show qualitative results in Figure 4.3. One main reason the PSNR of c-Glow is lower than

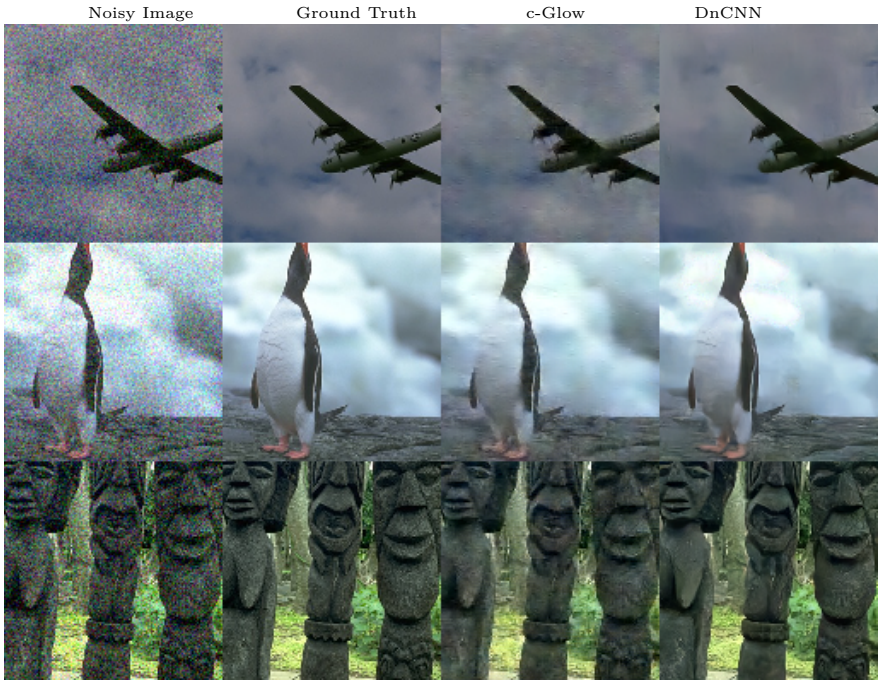


Figure 4.3: Example qualitative results.

DnCNN is that the images generated by DnCNN are smoother than the images generated by c-Glow. We believe this is caused by one drawback of flow-based models. Flow-based models use squeeze layers to fold input tensors to exploit the local correlation structure of an image. The squeeze layers use a spatial pixel-wise checkerboard mask to split the input tensor, which may cause values of neighbor pixels to vary non-smoothly.

4.3.5 Denoising for Depth Refinement

In this set of experiments, we use the seven scenes dataset [79], which contains noisy depth maps of natural scenes. The task is to denoise the depth maps. We use the same method as Wang et al. (2016) to process the dataset. We train our model on 200 images from the Chess scene and test on 5,500 images from other scenes. The images are randomly cropped to 96×128 pixels. We use PSNR as the metric. We compare c-Glow with ProximalNet [109], FilterForest [96], and BM3D [19]. For c-Glow, the parameters are set to be $L = 3, K =$

$8, n_c = 8$, and $n_w = 32$. Note that we use smaller conditioning networks for this task, because the images for this task are one-dimensional grayscale images.

We list the metric scores in Table 4.4. ProximalNet is a deep energy-based structured prediction model, and FilterForest and BM3D are traditional filter-based models. ProximalNet works better than filter-based baselines, and c-Glow gets a slightly better PSNR.

Table 4.4: Depth refinement scores (PSNR).

c-Glow	**ProximalNet**	**FilterForest**	**BM3D**
36.53	36.31	35.63	35.46

4.3.6 Image Inpainting

Inferring parts of images that are censored or occluded requires modeling of the structure of dependencies across pixels. In this set of experiments, we test c-Glow on the task of inpainting censored images from the CelebA dataset [65], which has around 200,000 images of faces. We randomly select 2,000 images as our test set. We centrally crop the images and resize them to 64×64 pixels. We use central block masks such that 25% of the pixels are hidden from the input. For c-Glow, we set $L = 3, K = 8, n_c = 64$, and $n_w = 128$. For training the model, we set the features x to be the occluded images and the labels y to be the center region that needs to be inpainted. We compare our method with DCGAN inpainting (DCGANi) [118], which is the state-of-the-art deep model for image inpainting. We use PSNR as our metric.

Table 4.5: Image inpainting scores (PSNR).

c-Glow	**DCGANi-b**	**DCGANi**
24.88	23.65	22.73

Figure 4.4: Sample results of c-Glow and DCGAN inpainting.

The PSNR scores are in Table 4.5. "DCGANi-b" represents DCGANi with Poisson blending. Figure 4.4 contains sample inpainting results. C-Glow outperforms DCGAN inpainting in both the PSNR scores and the quality of generated images. Note that the DCGAN inpainting method largely depends on postprocessing the images with Poisson blending, which can make the color of the inpainted region align with the surrounding pixels. However, the shapes of features like noses and eyes are still not well recovered. Even though the images inpainted by c-Glow are slightly darker than the original images, the shapes of features are well captured.

4.3.7 Discussion

We evaluated c-Glow on five different structured prediction tasks. Two tasks require discrete outputs (binary and multi-class segmentation) while the other three tasks require continuous variables. C-Glow works well on all the tasks and scores comparably to the best method for each task. We compare c-Glow with different baselines for each task, some specifically designed for that task and some that are general deep energy-based models. Our results show that c-Glow outperforms deep energy-based models on many tasks, e.g., scoring higher than DVN and NLStruct on binary segmentation. C-Glow also outperforms some deep models on some tasks, e.g., DCGAN inpainting. However, c-Glow's generated images are not smooth enough, so its PSNR scores are slightly below DnCNN and BM3D for denoising. C-Glow handles these different tasks with the same CN architecture with only slight changes to the size of latent networks, demonstrating c-Glow to be a strong general-purpose model.

4.4 Conclusion

In this chapter, we propose conditional generative flows (c-Glow), which are conditional generative models for structured output learning. The model allows the change-of-variables formula to transform conditional likelihood for high-dimensional variables. We show how to convert the Glow model to a conditional form by incorporating conditioning networks. Our model can train by directly maximizing exact likelihood, so it does not need surrogate objectives or approximate inference. With a learned model, we can efficiently draw conditional samples from the exact learned distribution. Our experiments test c-Glow on five structured prediction tasks, finding that c-Glow generates accurate conditional samples and has predictive abilities comparable to recent deep structured prediction approaches. In the future, we will develop different variants of c-Glow for more complicated tasks in NLP, and test them on different datasets.

Chapter 5

Weakly Supervised Label Learning Flows

Machine learning has achieved great success in many supervised learning tasks. However, in practice, data labeling is usually human intensive and costly. To address this problem, practitioners are turning to weakly supervised learning [128], which trains machine learning models with only noisy labels that are generated by human specified rules or pretrained models for related tasks. Recent research shows that these models trained with weak supervisions can also perform well.

Many existing weakly supervised learning methods [4, 8, 89, 91] learn a deterministic function that estimates the unknown labels $\mathbf{y}$ given input data $\mathbf{x}$ and weak signals $\mathbf{Q}$. Since the observed information is incomplete, the predictions based on it can be varied. However, these methods ignore this uncertainty between $\mathbf{x}$ and $\mathbf{y}$. In this paper, we develop *label learning flows* (LLF), a general framework for weakly supervised learning problems. LLF is a flow-based generative model [21, 22, 51, 92]. The main idea behind LLF is that we define the relationship between $\mathbf{x}$ and $\mathbf{y}$ with a probability distribution $p(\mathbf{y}|\mathbf{x})$, which is modeled by a conditional flow. In training, we use the weak signals $\mathbf{Q}$ to define a constrained space for $\mathbf{y}$ and then optimize the likelihood of all possible $\mathbf{y}$ that are within this constrained space. Therefore, this model captures all possible relationships between the input $\mathbf{x}$ and output $\mathbf{y}$. Learning LLF can be defined as a constrained optimization problem. We develop a

learning method for LLF that trains the conditional flow inversely and avoids estimating the labels. For prediction, we use the same sample-based method as c-Glow, i.e., Equation 4.2, to estimate the labels.

We apply LLF to three weakly supervised learning problems: weakly supervised classification [4, 75], weakly supervised regression, and unpaired point cloud completion [18, 114]. These three problems have very different label types and weak signals. Our method outperforms all other state-of-the-art methods on weakly supervised classification and regression, and it can perform comparably to other recent methods on unpaired point cloud completion. These experiments show that LLF is versatile and powerful.

5.1 Weakly Supervised Label Learning Flows

In this section, we introduce the label learning flows (LLF) framework for weakly supervised learning. Given $\mathbf{Q}$ and $\mathbf{x}$, we define a set of constraints to restrict the predicted label $\mathbf{y}$. These constraints can be inequalities, formatting like $c(\mathbf{x}, \mathbf{y}, \mathbf{Q}) \leq b$, or equations, formatting like $c(\mathbf{x}, \mathbf{y}, \mathbf{Q}) = b$. For simplicity, we represent this set of constraints as $\mathbf{C}(\mathbf{x}, \mathbf{y}, \mathbf{Q})$. Let Ω be the constrained space of all possible $\mathbf{y}$ defined by $\mathbf{C}(\mathbf{x}, \mathbf{y}, \mathbf{Q})$. Previous methods [4, 8, 89, 91] only look for one possible $\mathbf{y}$ within Ω. In contrast, LLF optimizes the conditional log-likelihood of all possible $\mathbf{y}$ within Ω, resulting in the following objective

$$\max_{\phi} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\mathbf{y} \sim U(\Omega)} \left[\log p(\mathbf{y}|\mathbf{x}, \phi) \right], \tag{5.1}$$

where $U(\Omega)$ is a uniform distribution within Ω, and $p(\mathbf{y}|\mathbf{x})$ is a continuous density model.

Let $\hat{\mathbf{y}}$ be the true label. We assume that each data point $\mathbf{x}_i$ only has one unique label $\hat{\mathbf{y}}_i$, so that $p_{\text{data}}(\mathbf{x}, \hat{\mathbf{y}}) = p_{\text{data}}(\mathbf{x})$. Let $q(\hat{\mathbf{y}}|\mathbf{x})$ be a certain model of $\hat{\mathbf{y}}$. Traditional supervised

learning learns a $q(\hat{\mathbf{y}}|\mathbf{x})$ that maximizes the cross entropy $\mathbb{E}_{p_{\text{data}}(\mathbf{x},\hat{\mathbf{y}})}[\log q(\hat{\mathbf{y}}|\mathbf{x},\phi)]$. Following theorem indicates that maximizing $\log p(\mathbf{y}|\mathbf{x})$ can be interpreted as maximizing a lower bound of $\log q(\hat{\mathbf{y}}|\mathbf{x})$.

Theorem 5.1. *Let $\Omega^* \subseteq \Omega$ is a tight enough space satisfying that $\hat{\mathbf{y}} \in \Omega^*$ and for two different $\hat{\mathbf{y}}_i$ and $\hat{\mathbf{y}}_j$, the Ω_i^* and Ω_j^* are non-overlapped. The volume of Ω^* is bounded such that $\frac{1}{|\Omega^*|} \leq M$. The relationship between $p(\mathbf{y}|\mathbf{x})$ and $q(\hat{\mathbf{y}}|\mathbf{x})$ can be defined as: $q(\hat{\mathbf{y}}|\mathbf{x}) = \int_{\mathbf{y}\in\Omega^*} p(\mathbf{y}|\mathbf{x})d\mathbf{y}$. Then maximizing $\log p(\mathbf{y}|\mathbf{x})$ can be interpreted as maximizing the lower bound of $\log q(\hat{\mathbf{y}}|\mathbf{x})$. That is,*

$$\mathbb{E}_{p_{data}(\mathbf{x})}\mathbb{E}_{\mathbf{y}\sim U(\Omega^*)}[\log p(\mathbf{y}|\mathbf{x},\phi)] \leq M\mathbb{E}_{p_{data}(\mathbf{x},\hat{\mathbf{y}})}[\log q(\hat{\mathbf{y}}|\mathbf{x},\phi)] \tag{5.2}$$

The complete proof is in the appendix. Theorem 5.1 shows that, when Ω is well-defined, learning LLF is analogous to dequantization [37, 103], i.e., a commonly used technique for generative models that converts a discrete variable to continuous by adding continuous noise to it. That is, LLF optimizes the likelihood of dequantized true labels. Optimizing Equation 5.1 will also optimize the certain model on true labels. In practice, the real constrained space, i.e., Ω, may be loose and does not fulfill the conditions in Theorem 5.1, which will result in inevitable errors that come from the weakly supervised setting. Moreover, for some regression problems, the ideal Ω^* only contains a single point: the ground truth label, i.e., $\Omega^* = \{\hat{\mathbf{y}}\}$.

5.1.1 Learning and Prediction

Training the model with Equation 5.1 requires sampling $\mathbf{y}$ within Ω. Using traditional sampling methods, e.g., uniform sampling, to sample $\mathbf{y}$ is inefficient. Due to the high dimensionality of sample space, the rejection rate would be prohibitively high. Moreover, estimating $\mathbf{y}$ at each iteration would result in an EM-like algorithm, which complicates the

training process. This problem can instead be solved with the invertibility of normalizing flows. That is, we rewrite $\log p(\mathbf{y}|\mathbf{x})$ as

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{x}) &= \log p_Z(\mathbf{f}_{\mathbf{x},\phi}(\mathbf{y})) + \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{f}_{\mathbf{x},\phi_i}}{\partial \mathbf{r}_{i-1}} \right) \right| \\ &= \log p_Z(\mathbf{z}) - \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{g}_{\mathbf{x},\phi_i}}{\partial \mathbf{r}_i} \right) \right|, \end{aligned} \tag{5.3}$$

where $\mathbf{g}_{\mathbf{x},\phi_i} = f^{-1}_{\mathbf{x},\phi_i}$ is the inverse flow.

With the inverse flow, Equation 5.1 can be interpreted as a constrained optimization problem

$$\begin{aligned} &\max_{\phi} \mathbb{E}_{p_{data}(\mathbf{x})} \mathbb{E}_{p_Z(\mathbf{z})} \left[\log p_Z(\mathbf{z}) - \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{g}_{\mathbf{x},\phi_i}}{\partial \mathbf{r}_i} \right) \right| \right], \\ &\text{s.t.} \quad \mathbf{C}(\mathbf{x}, \mathbf{g}_{\mathbf{x},\phi}(\mathbf{z}), \mathbf{Q}). \end{aligned} \tag{5.4}$$

In Equation 5.4, the original constraint for $\mathbf{z}$ is $\mathbf{g}_{\mathbf{x},\phi}(\mathbf{z}) \in \Omega$, and this constraint can be replaced with $\mathbf{C}(\mathbf{x}, \mathbf{g}_{\mathbf{x},\phi}(\mathbf{z}), \mathbf{Q})$. With this equation, the problem of sampling $\mathbf{y}$ within Ω is converted to sampling $\mathbf{z}$ from $p_Z(\mathbf{z})$, so that can be easily solved with an inverse flow. For efficient training, this constrained optimization problem can be approximated with the penalty method, resulting in the objective

$$\max_{\phi} \mathbb{E}_{p_{data}(\mathbf{x})} \mathbb{E}_{p_Z(\mathbf{z})} \left[\log p_Z(\mathbf{z}) - \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{g}_{\mathbf{x},\phi_i}}{\partial \mathbf{r}_i} \right) \right| - \lambda \mathbf{C}_r(\mathbf{x}, \mathbf{g}_{\mathbf{x},\phi}(\mathbf{z}), \mathbf{Q}) \right], \tag{5.5}$$

where λ is the penalty coefficient, and $\mathbf{C}_r()$ means we reformulate the constraints to be penalty losses. For example, an inequality constraint will be redefined as a hinge loss. In training, the inverse flow, i.e., $\mathbf{g}_{\mathbf{x},\phi}(\mathbf{z})$ estimates $\mathbf{y}$ and computes the likelihood simultaneously, removing the need of EM-like methods and making the training more straightforward.

In practice, the expectation with respect to $p_Z(\mathbf{z})$ can be approximated with Monte Carlo

estimate with L_t samples. Since we only need to obtain stochastic gradients, we follow previous works [50] and set $L_t = 1$.

Given a trained model, prediction requires outputting a label $\mathbf{y}_i$ for a data point $\mathbf{x}_i$. We use the same method as c-Glow, i.e., Equation 4.2 to estimate labels with sample average.

5.2 Case Study

In this section, we illustrate three scenarios of using LLF to address weakly supervised learning problems.

5.2.1 Weakly Supervised Classification

We follow previous works [4, 75] and consider binary classification. For each example, the label $\mathbf{y}$ is a two-dimensional vector within a one-simplex. That is, the $\mathbf{y} \in \mathcal{Y} = \{\mathbf{y} \in [0,1]^2 : \sum_j \mathbf{y}^{[j]} = 1\}$, where $\mathbf{y}^{[j]}$ is the jth dimension of $\mathbf{y}$. Each ground truth label $\hat{\mathbf{y}} \in \{0,1\}^2$ is a two-dimensional one-hot vector. We have M weak labelers, which will generate M weak signals for each data point $\mathbf{x}_i$, i.e., $\mathbf{q}_i = [\mathbf{q}_{i,1}, ..., \mathbf{q}_{i,M}]$. Each weak signal $\mathbf{q}_{i,m} \in \mathcal{Q} = \{\mathbf{q} \in [0,1]^2 : \sum_j \mathbf{q}^{[j]} = 1\}$ is a soft labeling of the data. In practice, if a weak labeler m fails to label a data point $\mathbf{x}_i$, the $\mathbf{q}_{i,m}$ can be null, i.e., $\mathbf{q}_{i,m} = \emptyset$ [3]. Following Arachie and Huang [4], we assume we have access to error rate bounds of these weak signals $\mathbf{b} = [\mathbf{b}_1, .., \mathbf{b}_M]$. These error rate bounds can be estimated based on empirical data [4], or set as constants [3]. Therefore, the weak signals imply constraints

$$\sum_{i=1, \mathbf{q}_{i,m} \neq \emptyset}^{N} (1 - \mathbf{y}_i^{[j]})\mathbf{q}_{i,m}^{[j]} + \mathbf{y}_i^{[j]}(1 - \mathbf{q}_{i,m}^{[j]}) \leq N_m \mathbf{b}_m^{[j]} \quad \forall m \in \{1, ..., M\}, \quad \forall j \in \{0,1\} \tag{5.6}$$

where N_m is the number of data points that are labeled by weak labeler m.

This problem can be solved with LLF, i.e., Equation 5.4, by defining $\mathbf{C}(\mathbf{x}, \mathbf{g}_{\mathbf{x},\phi}(\mathbf{z}, \mathbf{Q})$ to be a combination of weak signal constraints, i.e., Equation 5.6, and simplex constraints, i.e., $\mathbf{y} \in \mathcal{Y}$. The objective function of LLF for weakly supervised classification is

$$\begin{aligned}\max_{\phi} \quad & \log p_Z(\mathbf{z}) - \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{g}_{\mathbf{x},\phi_i}}{\partial \mathbf{r}_i} \right) \right| \\ & -\lambda_1 \left[\mathbf{g}_{\mathbf{x},\phi}(\mathbf{z}) \right]_+^2 - \lambda_2 \left[1 - \mathbf{g}_{\mathbf{x},\phi}(\mathbf{z}) \right]_+^2 - \lambda_3 \left(\sum_i \mathbf{g}_{\mathbf{x},\phi}(\mathbf{z})[i] - 1 \right)^2 \\ & -\lambda_4 \sum_{j=0}^{1} \sum_{m=1}^{M} \left[\sum_{i=0, \mathbf{q}_{i,m} \neq \emptyset}^{N} (1 - \mathbf{g}_{\mathbf{x},\phi}(\mathbf{z})_i^{[j]}) \mathbf{q}_{i,m}^{[j]} + \mathbf{g}_{\mathbf{x},\phi}(\mathbf{z})_i^{[j]} (1 - \mathbf{q}_{i,m}^{[j]}) - N_m \mathbf{b}_m^{[j]} \right]_+^2 \end{aligned} \tag{5.7}$$

where the second row describes the simplex constraints, and the last row is the weak signal constraints reformulated from Equation 5.6. The $[.]_+$ is a hinge function that returns its input if positive and zero otherwise. We omit the expectation terms for simplicity.

5.2.2 Weakly Supervised Regression

For weakly supervised regression, we predict one-dimensional continuous labels $y \in [0, 1]$ given input dataset $\mathcal{D} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and weak signals $\mathbf{Q}$. We define the weak signals as follows. For the mth feature of input data, we have access to a threshold ϵ_m, which splits $\mathcal{D}$ to two parts, i.e., $\mathcal{D}_1, \mathcal{D}_2$, such that for each $\mathbf{x}_i \in \mathcal{D}_1$, the $\mathbf{x}_{i,m} \geq \epsilon_m$, and for each $\mathbf{x}_j \in \mathcal{D}_2$, the $\mathbf{x}_{j,m} < \epsilon_m$. We also have access to estimated values of labels for subsets $\mathcal{D}_1$ and $\mathcal{D}_2$, i.e., $b_{m,1}$ and $b_{m,2}$. This design of weak signals tries to mimic that in practical scenarios, human experts can design rule-based methods for predicting labels for given data. For example, marketing experts can predict the prices of houses based on their size. For houses whose size is greater than a threshold, an experienced expert would know an estimate of their

average price. Assuming that we have M rule-based weak signals, the constraints can be mathematically defined as follows:

$$\frac{1}{|\mathcal{D}_{m,1}|}\sum_{i\in\mathcal{D}_{m,1}} y_i = b_{m,1}, \qquad \frac{1}{|\mathcal{D}_{m,2}|}\sum_{j\in\mathcal{D}_{m,2}} y_j = b_{m,2}, \qquad m \in 1, ..., M. \tag{5.8}$$

Plugging in Equation 5.8 to Equation 5.5, we have

$$\begin{aligned}\max_{\phi} \quad & \log p_Z(z) - \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{g}_{\mathbf{x},\phi_i}}{\partial r_i} \right) \right| - \lambda_1 \left[\mathbf{g}_{\mathbf{x},\phi}(z) \right]_+^2 - \lambda_2 \left[1 - \mathbf{g}_{\mathbf{x},\phi}(z) \right]_+^2 \\ & -\lambda_3 \sum_{m=1}^{M} \left(\frac{1}{|\mathcal{D}_{m,1}|} \sum_{i\in\mathcal{D}_{m,1}} \mathbf{g}_{\mathbf{x},\phi}(z)_i - b_{m,1} \right)^2 + \left(\frac{1}{|\mathcal{D}_{m,2}|} \sum_{j\in\mathcal{D}_{m,2}} \mathbf{g}_{\mathbf{x},\phi}(z)_j - b_{m,2} \right)^2,\end{aligned} \tag{5.9}$$

where the first two constraints restrict $y \in [0, 1]$, and the last term is the weak signal constraints reformulated from Equation 5.8.

5.2.3 Unpaired Point Cloud Completion

Unpaired point cloud completion [18, 114] is a practical problem in 3D scanning. Given a set of partial point clouds $\mathcal{X}_p = \{\mathbf{x}_1^{(p)}, ..., \mathbf{x}_N^{(p)}\}$, and a set of complete point clouds $\mathcal{X}_c = \{\mathbf{x}_1^{(c)}, ..., \mathbf{x}_N^{(c)}\}$, we want to restore each $\mathbf{x}_i^{(p)} \in \mathcal{X}_p$ by generating its corresponding clean and complete point cloud. Each point cloud is a set of points, i.e., $\mathbf{x}_i = \{\mathbf{x}_{i,1}, ..., \mathbf{x}_{i,T}\}$, where each $\mathbf{x}_{i,t} \in \mathcal{R}^3$ is a 3D point, and the counts T represent the number of points in a point cloud.

Note that the point clouds in $\mathcal{X}_p$ and $\mathcal{X}_c$ are *unpaired*, so directly modeling the relationship between $\mathbf{x}^{(c)}$ and $\mathbf{x}^{(p)}$ is impossible. This problem can be interpreted as an inexact supervised

learning problem [128], in which the weak supervision is given by the referred complete point clouds $\mathcal{X}_c$. We predict complete point clouds $\mathbf{y} \in \mathcal{Y}$ for partial point clouds in $\mathcal{X}_p$. The conditional distribution $p(\mathbf{y}|\mathbf{x}_p)$ is an exchangable distribution. We follow previous works [54, 117] and use De Finetti's representation theorem and variational inference to compute its lower bound as the objective.

$$\log p(\mathbf{y}|\mathbf{x}_p) \quad \geq \quad \mathbb{E}_{q(\mathbf{u}|\mathbf{x}_p)} \left[\sum_{i=1}^{T_c} \log p(\mathbf{y}_i|\mathbf{u}, \mathbf{x}_p) \right] - \mathrm{KL}(q(\mathbf{u}|\mathbf{x}_p)||p(\mathbf{u})), \tag{5.10}$$

where $q(\mathbf{u}|\mathbf{x}_p)$ is a variational distribution of latent variable $\mathbf{u}$. In practice, it can be represented by an encoder, and uses the reparameterization trick [50] to sample $\mathbf{u}$. The $p(\mathbf{u})$ is a standard Gaussian prior. The $p(\mathbf{y}_i|\mathbf{u}, \mathbf{x}_p)$ is defined by a conditional flow. We follow Chen et al. [18], Wu et al. [114] and use the adversarial loss and Hausdorff distance loss as constraints. The final objective function is

$$\begin{aligned} \max_{\phi} \quad & \mathbb{E}_{q(\mathbf{u}|\mathbf{x}_p)} \left[\sum_{t=1}^{T_c} \log p_Z(\mathbf{z_t}) - \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial \mathbf{g}_{\mathbf{u},\mathbf{x}_p,\phi_i}}{\partial \mathbf{r}_{t,i}} \right) \right| \right] - \mathrm{KL}(q(\mathbf{u}|\mathbf{x}_p)||p(\mathbf{u})) \\ & -\mathbb{E}_{q(\mathbf{u}|\mathbf{x}_p)} \left[\lambda_1 (D(\mathbf{g}_{\mathbf{u},\mathbf{x}_p,\phi}(\mathbf{z})) - 1)^2 - \lambda_2 d_H(\mathbf{g}_{\mathbf{u},\mathbf{x}_p,\phi}(z), \mathbf{x}_p) \right], \end{aligned} \tag{5.11}$$

where $D()$ represents the discriminator of a least square GAN [73]. It is trained with the referred complete point clouds, so that contains their structure and geometric information. It provides a score for each generated complete point cloud indicating its quality and fidelity, i.e., how close it is to real complete point clouds. The $d_H()$ represents the Haudorsff distance, which measures the distance between a generated complete point cloud and its corresponding input partial point cloud. For clarity, we use $\mathbf{z}_t$ and $\mathbf{r}_t$ to represent variables of the tth point in a point cloud, and $\mathbf{g}_{\mathbf{u},\mathbf{x}_p,\phi}(\mathbf{z})$ to represent a generated point cloud. Detailed derivations of Equation 5.11 are in the appendix.

Training with Equation 5.11 is different from the previous settings, because we also need to

train the discriminator of the GAN. The objective for $D()$ is

$$\min_{D} \mathbb{E}_{p_{data}(\mathbf{x}_c)} \left[(D(\mathbf{x}_c) - 1)^2 \right] + \mathbb{E}_{p_{data}(\mathbf{x}_p), p_Z(\mathbf{z}), q(\mathbf{u}|\mathbf{x}_p)} \left[D(\mathbf{g}_{\mathbf{x}_p, \mathbf{u}, \phi}(\mathbf{z}))^2 \right]. \tag{5.12}$$

The training process is similar to traditional GAN training. The inverse flow $\mathbf{g}_{\mathbf{u}, \mathbf{x}_p, \phi}$ can be roughly seen as the generator. In training, we train the flow to optimize Equation 5.11 and the discriminator to optimize Equation 5.12, alternatively.

5.3 Related Work

In this section, we introduce the research that most related to LLF.

5.3.1 Weakly Supervised Learning

For weakly supervised classification, we use the same strategy as adversarial label learning (ALL) [4] to define constraint functions based on weak signals. ALL then uses a min-max optimization to learn the model parameters and estimate $\mathbf{y}$ alternatively. In contrast to ALL, LLF is a generative model, so it can learn the model parameters and output $\mathbf{y}$ simultaneously, and it does not need a min-max optimization. Moreover, LLF optimizes the likelihoods of all possible $\mathbf{y}$s within Ω, while ALL only estimates one possible $\mathbf{y}$. Other methods also constrain the label space of the predicted labels using weak supervision [3, 4, 74, 75]. These methods are deterministic and developed for classification tasks. However, LLF can be used for other weakly supervised learning tasks and uses sampling during inference.

Non-constraint based weak supervision methods typically assume a joint distribution for the weak signals and the true labels of the data. These methods use a latent variable model to

estimate the labels of the data while accounting for the dependency among the weak signals [28, 90, 91]. Like these methods, we assume a family of distributions for the label space of the data. This space is defined by the constraints of the weak supervision and the data. Unlike these methods, we use a flow network rather than a graphical model to solve for the label of the data. Additionally, we do not solve for the dependence amongst the weak signals thereby avoiding the need for making extra assumptions.

5.3.2 Normalizing Flows

Normalizing flows [21, 22, 51, 92] have gained recent attention because of their advantages of exact latent variable inference and log-likelihood evaluation. Specifically, conditional normalizing flows have been widely applied to many supervised learning problems [67, 68, 86, 105] and semi-supervised classification [7, 43]. However, normalizing flows have not previously been applied to weakly supervised learning problems.

Our inverse training method for LLF is similar to that of injective flows [58]. Injective flows are used to model unconditional datasets. They use an encoder network to map the input data $\mathbf{x}$ to latent code $\mathbf{z}$, and then they use an inverse flow to map $\mathbf{z}$ back to $\mathbf{x}$, resulting in an autoencoder architecture. Different from injective flow, LLF directly samples $\mathbf{z}$ from a prior distribution and uses a conditional flow to map $\mathbf{z}$ back to $\mathbf{y}$ conditioned on $\mathbf{x}$. We use constraint functions to restrict $\mathbf{y}$ to be valid, so that does not need an encoder network.

5.3.3 Point Cloud Modeling

Recently, Yang et al. [117] and Tran et al. [104] combine normalizing flows with variational autoencoders [50] and developed continuous and discrete normalizing flows for point clouds. The basic idea of point normalizing flows is to use a conditional flow to model each point

in a point cloud. The conditional flow is conditioned on a latent variable generated by an encoder. To guarantee exchangeability, the encoder uses a PointNet [87] to extract features from input point clouds.

The unpaired point cloud completion problem is first defined by [18]. They develop pcl2pcl—a GAN [31] based model—to solve it. Their method is two-staged. In the first stage, it trains autoencoders to map partial and complete point clouds to their latent space. In the second stage, a GAN is used to transform the latent features of partial point clouds to latent features of complete point clouds. In their follow-up paper [114], they develop a variant of pcl2pcl, called multi-modal pcl2pcl (mm-pcl2pcl), which incorporates random noise to the generative process, so that can capture the uncertainty in reasoning.

In contrast to pcl2pcl, LLF can be trained end-to-end. When applying LLF to this problem, LLF has a similar framework to VAE-GAN [61]. The main differences are that LLF models a conditional distribution of points, and its encoder is a point normalizing flow.

5.4 Empirical Study

In this section, we evaluate LLF on the three weakly supervised learning problems.

Model architecture. For weakly supervised classification and unpaired point cloud completion, the labels $\mathbf{y}$ are multi-dimensional variables. We follow [54] and use flows with only conditional coupling layers. We use the same method as [54] to define the conditional affine layer. Each flow model contains 8 flow steps. For unpaired point cloud completion, each flow step has 3 coupling layers. For weakly supervised classification, each flow step has 2 coupling layers. For weakly supervised regression, since y is a scalar, we use simple conditional affine transformation as flow layer, which is defined as:$y = \mathbf{s}(\mathbf{x}) * z + \mathbf{b}(\mathbf{x})$, where $\mathbf{s}$ and $\mathbf{b}$ are two

neural networks that take $\mathbf{x}$ as input and output parameters for y. The flow for this problem contains 8 conditional affine transformations.

For the unpaired point cloud completion task, we need to also use an encoder network, i.e., $q(\mathbf{u}|\mathbf{x}_p)$ and a discriminator $D()$. We follow [54, 114] and use PointNet [87] in these two networks to extract features for point clouds.

Experiment setup. In weakly supervised classification and regression experiments, we assume that the ground truth labels are inaccessible, so tuning hyper-parameters for models are impossible. We use default settings for all hyper-parameters of LLF, e.g., λs and learning rates. We fix $\lambda = 10$ and use Adam [49] with default settings, i.e., $\eta = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We use an exponential learning rate scheduler with a decreasing rate of 0.996 to guarantee convergence. We track the decrease of loss and when the decrease is small enough, the training stops. Following previous works [3, 4], we use full gradient optimization to train the models. For fair comparison, we run each experiment 5 times with different random seeds $\{0, 10, 100, 123, 1234\}$.

For experiments with unpaired point cloud completion, we tune the hyper-parameters using validation sets. We use Adam with an initial learning rate $\eta = 0.0001$ and default βs. The best coefficients for the constraints in Equation 5.11 are $\lambda_1 = 10, \lambda_2 = 100$. We use stochastic optimization to train the models, and the batch size is 32. Each model is trained for at most 2000 epochs.

5.4.1 Weakly Supervised Classification

Datasets. We follow Arachie and Huang [3, 4] and conduct experiments on 12 datasets. Specifically, the Breast Cancer, OBS Network, Cardiotocography, Clave Direction, Credit Card, Statlog Satellite, Phishing Websites, Wine Quality are tabular datasets from the UCI

repository [24]. The Fashion-MNIST [115] is an image set with 10 classes of clothing types. We choose 3 pairs of classes, i.e., dresses/sneakers (DvK), sandals/ankle boots (SvA), and coats/bags (CvB), to conduct binary classification. We follow [4] and create 3 synthetic weak signals for each dataset. Each dataset is split to training set, simulation set and test set. The error rate bounds are estimated based on the simulation set. The IMDB [71], SST [99] and YELP are real text datasets. We follow [3] and use keyword-based weak supervision. Each dataset has more than 10 weak signals. The error rate bounds are set as 0.01. For experiments on tabular datasets, we set the maximum epochs to 2000. For experiments on real text datasets, we set the maximum epochs to 500.

Baselines. We compare our method with state-of-the-art methods for weakly supervised classification. For the experiments on tabular datasets and image sets, we use ALL [4], generalized expectation (GE) [23, 72] and averaging of weak signals (AVG). For experiments on text datasets, we use CLL [3], Snorkel MeTaL [90], Data Programming (DP) [91], regularized minimax conditional entropy for crowdsourcing (MMCE) [127], and majority-vote (MV). Note that some baselines, e.g., DP, CLL, used on text datasets are two-stage method. That is, they first predict labels for data points, and then use estimated labels to train downstream classifiers. For better comparison, we develop a two-stage variant of LLF, i.e., LLF-TS, which first infers the labels, and then train a classifier with these inferred labels as a final predictor. For two-stage methods, we follow [3] and use a two-layer MLP as the classifier. We also show supervised learning (SL) results for reference. More details about the experiment settings are in the appendix.

Results. We report the mean and standard deviation of accuracy on test sets in Table 5.1 and Table 5.2. For experiments on tabular and image datasets, LLF outperforms other baselines on 10/11 datasets. On some datasets, LLF can perform as well as supervised learning methods. For experiments on text datasets, LLF outperform other baselines on

2/3 datasets. LLF-TS performs slightly worse than LLF, we feel that one possible reason is LLF is a probabilistic model, which uses the average of samples as estimate labels, so that can slightly smooth out anomalous values, and improve predictions. These results prove that LLF is powerful and effective. In our experiments, we also found that the performance of LLF will also be impacted by different initialization of weights. This is why LLF has relatively larger variance on some datasets.

Table 5.1: Test set accuracy on tabular and image datasets. We report the mean accuracy of 5 experiments, and the subscripts are standard deviation. LLF outperforms other baselines on 10 datasets.

	LLF	ALL	GE	AVG	SL
Fashion MNIST (DvK)	$\mathbf{1.000_{0.000}}$	$0.995_{0.000}$	$0.979_{0.000}$	$0.835_{0.000}$	$1.000_{0.000}$
Fashion MNIST (SvA)	$\mathbf{0.944_{0.001}}$	$0.908_{0.000}$	$0.501_{0.000}$	$0.791_{0.000}$	$0.972_{0.000}$
Fashion MNIST (CvB)	$\mathbf{0.916_{0.038}}$	$0.805_{0.000}$	$0.501_{0.000}$	$0.740_{0.000}$	$0.988_{0.000}$
Breast Cancer	$\mathbf{0.968_{0.008}}$	$0.937_{0.019}$	$0.933_{0.016}$	$0.911_{0.023}$	$0.973_{0.007}$
OBS Network	$0.684_{0.006}$	$0.691_{0.011}$	$0.676_{0.010}$	$\mathbf{0.709_{0.024}}$	$0.704_{0.032}$
Cardiotocography	$\mathbf{0.931_{0.010}}$	$0.795_{0.011}$	$0.663_{0.061}$	$0.902_{0.047}$	$0.941_{0.008}$
Clave Direction	$\mathbf{0.858_{0.017}}$	$0.750_{0.013}$	$0.756_{0.028}$	$0.707_{0.003}$	$0.963_{0.001}$
Credit Card	$\mathbf{0.680_{0.022}}$	$0.678_{0.021}$	$0.492_{0.088}$	$0.602_{0.010}$	$0.717_{0.031}$
Statlog Satellite	$\mathbf{0.997_{0.002}}$	$0.959_{0.008}$	$0.987_{0.012}$	$0.915_{0.011}$	$0.999_{0.001}$
Phishing Websites	$\mathbf{0.906_{0.003}}$	$0.896_{0.005}$	$0.870_{0.009}$	$0.848_{0.002}$	$0.929_{0.001}$
Wine Quality	$\mathbf{0.647_{0.017}}$	$0.623_{0.000}$	$0.445_{0.014}$	$0.555_{0.000}$	$0.685_{0.000}$

Table 5.2: Test set accuracy on real text datasets. LLF outperforms other baselines on 2 datasets.

	LLF	LLF-TS	CLL	MMCE	DP	MV	MeTaL	SL
SST	$\mathbf{0.746_{0.003}}$	$0.729_{0.004}$	$0.729_{0.001}$	0.727	$0.720_{0.001}$	$0.720_{0.001}$	$0.728_{0.001}$	$0.792_{0.001}$
IMDB	$\mathbf{0.752_{0.001}}$	$0.750_{0.006}$	$0.740_{0.005}$	0.551	$0.623_{0.007}$	$0.724_{0.004}$	$0.742_{0.004}$	$0.820_{0.003}$
YELP	$0.811_{0.001}$	$0.786_{0.002}$	$\mathbf{0.840_{0.001}}$	0.680	$0.760_{0.005}$	$0.798_{0.007}$	$0.780_{0.002}$	$0.879_{0.001}$

Ablation Study. We can directly train the model using only the constraints as the objective function. In our experiments, we found that training LLF without likelihood (LLF-w/o-nll) will still work. However, the model performs worse than training with likelihood. We believe that this is because the likelihood helps accumulate more probability mass to the constrained

space Ω, so the model will more likely generate $\mathbf{y}$ samples within Ω, and the predictions are more accurate.

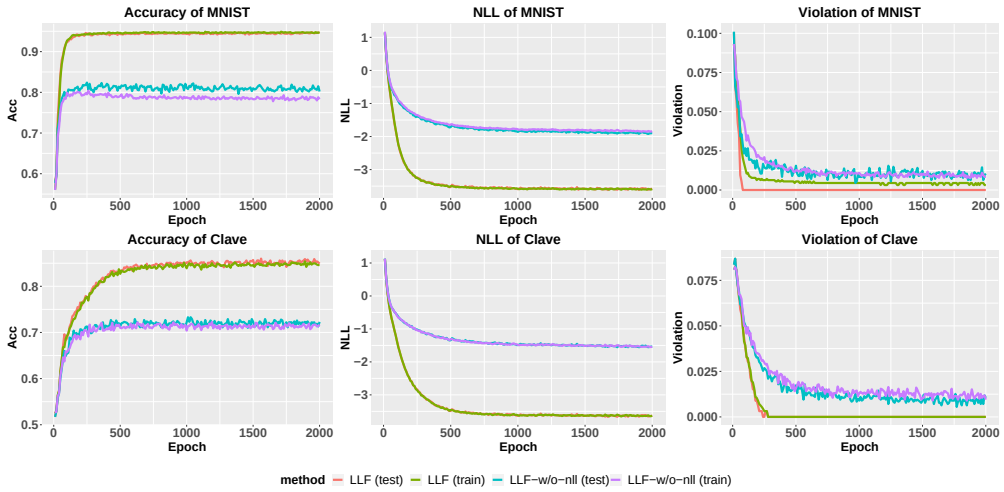


Figure 5.1: Evolution of accuracy, likelihood and violation of weak signal constraints. Training with likelihood makes LLF accumulate more probability mass to the constrained space, so that the generated $\mathbf{y}$ are more likely to be within Ω, and the predictions are more accurate.

5.4.2 Weakly Supervised Regression

Datasets. We use 3 tabular datasets from the UCI repository [24]: Air Quality, Temperature Forecast, and Bike Sharing dataset. For each dataset, we randomly choose 5 features to develop the rule based weak signals. We split each dataset to training, simulation, and test sets. The simulation set is then used to compute the threshold ϵs, and the estimated label values bs. Since we do not have human experts to estimate these values, we use the mean value of a feature as its threshold, i.e., $\epsilon_m = \frac{1}{|\mathcal{D}_{\text{valid}}|} \sum_{i \in \mathcal{D}_{\text{valid}}} \mathbf{x}_i[m]$. We then compute the estimated label values $b_{m,1}$ and $b_{m,2}$ based on labels in the valid set. Note that the labels in the simulation set are only used for generating weak signals, simulating human expertise. In training, we still assume that we do not have access to labels. The original label is within an interval $[l_y, u_y]$. We normalize the original label to within $[0, 1]$ by computing

$y = (y - l_y)/(u_y - l_y)$. In prediction, we recover the predicted label to original value by computing $y = y(u_y - l_y) + l_y$.

Baselines. To the best of our knowledge, there are no methods specifically designed for weakly supervised regression of this form. We use average of weak signals (AVG) and LFF-w/o-nll as baselines. We also report the supervised learning results for reference.

Results. We use root square mean error (RSME) as metric. The results of test set are in Table 5.3. In general, LLF can predict reasonable labels. Its results are much better than AVG or any of the weak signals alone. Similar to the classification results, training LLF without using likelihood will reduce its performance.

Table 5.3: Test set RMSE of different methods. The numbers in brackets indicate the label's range. LLF outperforms other baselines on all datasets.

	LLF	LLF-w/o-nll	AVG	SL
Air Quality (0.1847 $\sim$ 2.231)	$\mathbf{0.211_{0.009}}$	$0.266_{0.004}$	$0.373_{0.005}$	$0.123_{0.002}$
Temperature Forecast (17.4 $\sim$ 38.9)	$\mathbf{2.552_{0.050}}$	$2.656_{0.055}$	$2.827_{0.027}$	$1.465_{0.031}$
Bike Sharing (1 $\sim$ 999)	$\mathbf{157.348_{0.541}}$	$162.697_{1.585}$	$171.338_{1.300}$	$141.920_{1.280}$

5.4.3 Unpaired Point Cloud Completion

Datasets. We use the Partnet [77] dataset in our experiments. We follow [114] and conduct experiments on the 3 largest classes of PartNet: Table, Chair, and Lamp. We treat each class as a dataset, which is split to training, validation, and test sets based on official splits of PartNet. For each point cloud, we remove points of randomly selected parts to create a partial point cloud. We follow [18, 114] and let the partial point clouds have 1024 points, and the complete point clouds have 2048 points. We let the latent variable $\mathbf{u}$ of the VAE to be a 128-dimensional vector.

Metrics. We follow [114] and use minimal matching distance (MMD) [2], total mutual

difference (TMD), and unidirectional Hausdorff distance (UHD) as metrics. MMD measures the quality of generated. A lower MMD is better. TMD measures the diversity of samples. A higher TMD is better. UHD measures the fidelity of samples. A lower UHD is better.

Baselines. We compare our method with pcl2pcl [18], mm-pcl2pcl [114], and LLF-w/o-nll. We use two variants of mm-pcl2pcl. Another variant is called mm-pcl2pcl-im, which is different from the original model in that it jointly trains the encoder of modeling multi-modality and the GAN.

Table 5.4: Evaluation results on PartNet. LLF performs comparable to baselines.

PartNet	**Chair**			**Lamp**			**Table**		
	MMD↓	TMD↑	UHD↓	MMD↓	TMD↑	UHD↓	MMD↓	TMD↑	UHD↓
LLF	1.72	0.63	5.74	2.11	0.57	4.71	1.57	0.55	5.42
LLF-w/o-nll	1.79	0.47	5.49	2.21	0.41	**4.61**	1.57	0.43	5.13
pcl2pcl	1.90	0.00	**4.88**	2.50	0.00	4.64	1.90	0.00	**4.78**
mm-pcl2pcl	**1.52**	**2.75**	6.89	**1.97**	**3.31**	5.72	**1.46**	**3.30**	5.56
mm-pcl2pcl-im	1.90	1.01	6.65	2.55	0.56	5.40	1.54	0.51	5.38

Results. We list the test set results in Table 5.4. In general, pcl2pcl has the best fidelity, i.e., lowest UHD. This is because pcl2pcl is a discriminative model, and it will only predict one certain sample for each input. This is also why pcl2pcl has the worse diversity as measured by TMD. Mm-pcl2pcl has the best diversity. However, as shown in Figure 5.2, some samples generated by mm-pcl2pcl are invalid, i.e., they are totally different from the input partial point clouds. Therefore, mm-pcl2pcl has the worse fidelity. LLF scores between pcl2pcl and mm-pcl2pcl. It has better UHD than mm-pcl2pcl, and better TMD and MMD than pcl2pcl. The LLF-w/0-nll has a slightly better UHD than LLF. We believe this is because, without using the likelihood, LLF-w/o-nll is trained directly by optimizing the Hausdorff distance. However, the sample diversity and quality, i.e., TMD and MMD, are worse than LLF. As argued by [117], the current metrics for evaluating point cloud samples all have flaws, so these scores cannot be treated as hard metrics for evaluating model performance.

We visualize some samples in Figure 5.2 and in the appendix. These samples show that LLF can generate samples that comparable to mm-pcl2pcl.

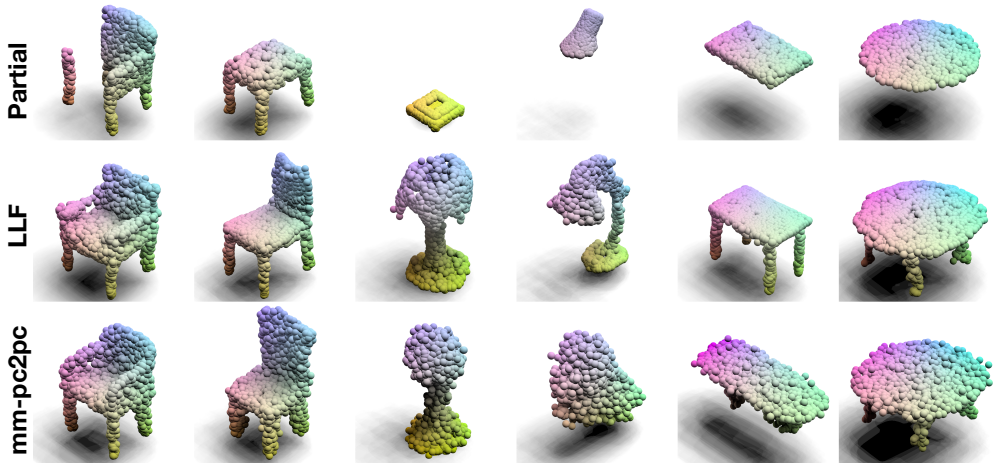


Figure 5.2: Random sample point clouds generated LLF and mm-pc2pc. The point clouds generated by LLF are as realistic as mm-pc2pc. Mm-pc2pc has a higher diversity in samples. However, sometimes it may generate unreasonable or invalid shapes.

5.5 Conclusion

In this chapter, we propose label learning flows, which represent a general framework for weakly supervised learning. LLF uses a conditional flow to define the conditional distribution $p(\mathbf{y}|\mathbf{x})$, so that can model the uncertainty between input $\mathbf{x}$ and all possible $\mathbf{y}$. Learning LLF is a constrained optimization problem that optimizes the likelihood of all possible $\mathbf{y}$ within the constrained space defined by weak signals. We develop a specific training method to train LLF inversely, avoiding the need of estimating $\mathbf{y}$. We apply LLF to three weakly supervised learning problems, and the results show that our method outperforms many state-of-the-art baselines on the weakly supervised classification and regression problems, and performs comparably to other new methods for unpaired point cloud completion.

Chapter 6

Conclusion and Future Work

Modeling structured data with machine learning models is an important problem. Deep generative models combine deep learning with probability theory, and have become a popular way to model complex datasets. Normalizing flows are a novel class of deep generative models, which have unique advantages of exact likelihood calculation and exact latent code inference. In this dissertation, we explore modeling complex and structured data with flow-based generative models, and focus on three types of machine learning problems, i.e., unsupervised learning, supervised learning and weakly supervised learning.

We first develop Woodbury transformations for general unsupervised normalizing flows, which use Woodbury matrix identity to compute the inverse transformations and Sylvester's determinant identity to compute Jacobian determinants. Woodbury transformations improve the flexibility and expressiveness of flow-based models while keep the computation efficiency, so that can scale well to high-dimensional variables. We test Woodbury layers on several image sets and compare them with other state-of-the-art flow layers. Our results show that Woodbury transformations can effectively improve the model performance, and scale well to high-resolution images.

We then develop c-Glow for structured output learning. C-Glow can be trained by directly maximizing the likelihoods, and does not need surrogate objectives or approximate inference. Therefore, training c-Glow is much more straightforward than other energy-based deep structured prediction models. We test c-Glow on five structured prediction tasks in

computer vision, and the results show that c-Glow can perform comparably well to other baselines.

Third, we develop label learning flows for weakly supervised learning. We interpret weakly supervised learning as a constrained optimization problem, and develop an inverse training method for LLF, which can train the conditional flows without sampled labels. We apply LLF to three weakly supervised learning problems, Experiment results indicate that LLF is effective and versatile.

Our works develop flow based models for different types of data, which prove that normalizing flows are powerful and can be applied to a variety of problems. In the future, we can combine these three methods and develop flow based models for weakly supervised structured prediction problems such as weakly supervised object detection [123]. This will require using Woodbury transformations to model dependencies of high-dimensional labels, defining conditional flows with the c-Glow architecture, and formulating the problem with LLF.

Aside from that, normalizing flows still have some drawbacks, which may be improved in the future. First, flow based models usually need more computational costs, and are challenging to scale. This is because each flow layer is a bijective function, and requires the input and output have the same dimensionality. Woodbury transformations use low rank matrices to factorize the original weight matrices, and can slightly alleviate this problem. Injective flows [58] relax this constraint by using injective flow layers, which allow the input and output to have different dimensionalities. However, injective flows can only approximate the likelihood, and need auxiliary encoder to assist training.

Second, normalizing flows now have a worse sample quality compared to state-of-the-art deep generative models, especially for high-dimensional data, e.g., high-resolution images. Therefore, they does not perform well on tasks related to sample generation, e.g., domain adapta-

tion [129]. Recent research shows that stochastic flow layers [112] can improve expressiveness of normalizing flows. Therefore, combining stochastic transformations of DDPMs [38] with deterministic flow layers can be a potential way to improve sample quality of flow based models.

Finally, many flow layers have been developed recently in order to improve the performance of flow based models. However, we still do not have a guidance for developing flow architectures. Currently, multi-scale architecture [22, 51] is a popular way to compose flow layers. In the future, we may develop intelligent architecture search methods [26] to automatically combine flow layers and form flow based models.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[2] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018.

[3] Chidubem Arachie and Bert Huang. Constrained labeling for weakly supervised learning. In *International Conference in Uncertainty in Artificial Intelligence*, 2021.

[4] Chidubem Arachie and Bert Huang. A general framework for adversarial label learning. *Journal of Machine Learning Research*, 22(118):1–33, 2021.

[5] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. on Pattern Analysis and Mach. Intell.*, 33(5):898–916, 2010.

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[7] Andrei Atanov, Alexandra Volokhova, Arsenii Ashukha, Ivan Sosnovik, and Dmitry Vetrov. Semi-conditional normalizing flows for semi-supervised learning. *arXiv preprint arXiv:1905.00505*, 2019.

BIBLIOGRAPHY

[8] Stephen H Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, et al. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 International Conference on Management of Data*, pages 362–375, 2019.

[9] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2018.

[10] David Belanger and Andrew McCallum. Structured prediction energy networks. In *Intl. Conf. on Machine Learning*, pages 983–992, 2016.

[11] David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structured prediction energy networks. In *Intl. Conf. on Machine Learning*, pages 429–439, 2017.

[12] Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

[13] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[14] Eran Borenstein and Shimon Ullman. Class-specific, top-down segmentation. In *European Conf. on Comp. Vision*, pages 109–122, 2002.

[15] Liang-Chieh Chen, Alexander Schwing, Alan Yuille, and Raquel Urtasun. Learning deep structured models. In *International Conference on Machine Learning*, pages 1785–1794, 2015.

[16] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural

ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[17] Tian Qi Chen, Jens Behrmann, David K Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*, pages 9913–9923, 2019.

[18] Xuelin Chen, Baoquan Chen, and Niloy J Mitra. Unpaired point cloud completion on real scans using adversarial training. *arXiv preprint arXiv:1904.00069*, 2019.

[19] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Trans. on Image Proc.*, 16(8):2080–2095, Aug 2007.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[21] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[22] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

[23] Gregory Druck, Gideon Mann, and Andrew McCallum. Learning from labeled features using generalized expectation criteria. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 595–602, 2008.

[24] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[25] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019.

[26] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[27] Marc Finz, Pavel Izmailov, Wesley Maddox, Polina Kirichenko, and Andrew Gordon Wilson. Invertible convolutional networks. In *ICML Workshop on Invertible Neural Networks and Normalizing Flows*, 2019.

[28] Daniel Fu, Mayee Chen, Frederic Sala, Sarah Hooper, Kayvon Fatahalian, and Christopher Ré. Fast and three-rious: Speeding up weak supervision with triplet methods. In *International Conference on Machine Learning*, pages 3280–3291. PMLR, 2020.

[29] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

[30] Zoubin Ghahramani, Sam Roweis, and NIPS Tutorial. Probabilistic models for unsupervised learning. *Gatsby Computational Neuroscience Unit University College London, NIPS Tutorials*, 1999.

[31] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[32] Colin Graber, Ofer Meshi, and Alexander Schwing. Deep structured prediction with nonlinear output transformations. In *Advances in Neural Information Processing Systems*, pages 6320–6331, 2018.

[33] Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[34] Michael Gygli, Mohammad Norouzi, and Anelia Angelova. Deep value networks learn to evaluate and iteratively refine structured outputs. In *Proc. of the Intl. Conf. on Machine Learning*, pages 1341–1351, 2017.

[35] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[36] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[37] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.

[38] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

[39] Emiel Hoogeboom, Rianne van den Berg, and Max Welling. Emerging convolutions for generative normalizing flows. *arXiv preprint arXiv:1901.11137*, 2019.

[40] Emiel Hoogeboom, Jorn WT Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. *arXiv preprint arXiv:1905.07376*, 2019.

[41] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779*, 2018.

[42] Gary B Huang, Vidit Jain, and Erik Learned-Miller. Unsupervised joint alignment of complex images. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[43] Pavel Izmailov, Polina Kirichenko, Marc Finzi, and Andrew Gordon Wilson. Semi-supervised learning with normalizing flows. In *International Conference on Machine Learning*, pages 4615–4630. PMLR, 2020.

[44] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

[45] Matthew J Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.

[46] Andrew Kae, Kihyuk Sohn, Honglak Lee, and Erik Learned-Miller. Augmenting CRFs with Boltzmann machine shape priors for image labeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2019–2026, 2013.

[47] Mahdi Karami, Dale Schuurmans, Jascha Sohl-Dickstein, Laurent Dinh, and Daniel Duckworth. Invertible convolutional flow. In *Advances in Neural Information Processing Systems*, pages 5636–5646, 2019.

[48] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

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

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

[51] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

[52] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

[53] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[54] Roman Klokov, Edmond Boyer, and Jakob Verbeek. Discrete point flow networks for efficient point cloud generation. *arXiv preprint arXiv:2007.10170*, 2020.

[55] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[56] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Advances in Neural Information Processing Systems*, pages 109–117, 2011.

[57] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[58] Abhishek Kumar, Ben Poole, and Kevin Murphy. Regularized autoencoders via relaxed injective probability flow. In *International Conference on Artificial Intelligence and Statistics*, pages 4292–4301. PMLR, 2020.

[59] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. *arXiv preprint arXiv:1703.01925*, 2017.

[60] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Intl. Conf. on Machine Learning*, 2001.

[61] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning*, pages 1558–1566. PMLR, 2016.

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

[63] Yang Li, Shoaib Akbar, and Junier B Oliva. Flow models for arbitrary conditional likelihoods. *arXiv preprint arXiv:1909.06319*, 2019.

[64] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. *Advances in neural information processing systems*, 31:7795–7804, 2018.

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

[66] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[67] You Lu and Bert Huang. Structured output learning with conditional generative flows. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5005–5012, 2020.

[68] Andreas Lugmayr, Martin Danelljan, Luc Van Gool, and Radu Timofte. Srflow: Learning the super-resolution space with normalizing flow. In *European Conference on Computer Vision*, pages 715–732. Springer, 2020.

[69] Xuezhe Ma, Xiang Kong, Shanghang Zhang, and Eduard Hovy. Macow: Masked convolutional generative flow. In *Advances in Neural Information Processing Systems*, pages 5891–5900, 2019.

[70] Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. Flowseq: Non-autoregressive conditional sequence generation with generative flow. *arXiv preprint arXiv:1909.02480*, 2019.

[71] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[72] Gideon S Mann and Andrew McCallum. Generalized expectation criteria for semi-supervised learning with weakly labeled data. *Journal of machine learning research*, 11(2), 2010.

[73] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[74] A. Mazzetto, C. Cousins, D. Sam, S. H. Bach, and E. Upfal. Adversarial multiclass learning under weak supervision with performance guarantees. In *International Conference on Machine Learning (ICML)*, 2021.

[75] Alessio Mazzetto, Dylan Sam, Andrew Park, Eli Upfal, and Stephen Bach. Semi-supervised aggregation of dependent weak supervision sources with performance guarantees. In *International Conference on Artificial Intelligence and Statistics*, pages 3196–3204. PMLR, 2021.

[76] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[77] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019.

[78] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Transactions on Graphics (TOG)*, 38(5):1–19, 2019.

[79] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, volume 11, pages 127–136, 2011.

[80] Mohammad Norouzi, Samy Bengio, Zhifeng Chen, Navdeep Jaitly, Mike Schuster, Yonghui Wu, and Dale Schuurmans. Reward augmented maximum likelihood for neural structured prediction. In *Advances in Neural Information Processing Systems*, pages 1723–1731, 2016.

[81] Sebastian Nowozin and Christoph H. Lampert. Structured learning and prediction

in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6: 185–365, 2011.

[82] Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3898–3907. PMLR, 2018.

[83] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[84] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[85] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

[86] Albert Pumarola, Stefan Popov, Francesc Moreno-Noguer, and Vittorio Ferrari. C-flow: Conditional generative flow models for images and 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7949–7958, 2020.

[87] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[88] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[89] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 11, page 269. NIH Public Access, 2017.

[90] Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. Training complex models with multi-task weak supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4763–4771, 2019.

[91] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29:3567–3575, 2016.

[92] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[93] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[94] Stefan Roth and Michael J Black. Fields of experts. *International Journal of Computer Vision*, 82(2):205, 2009.

[95] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115 (3):211–252, 2015.

[96] Sean Ryan Fanello, Cem Keskin, Pushmeet Kohli, Shahram Izadi, Jamie Shotton,

Antonio Criminisi, Ugo Pattacini, and Tim Paek. Filter forests for learning data-dependent convolutional kernels. In *Proc. of the IEEE Conf. on Computer Vis. and Pattern Recog.*, pages 1709–1716, 2014.

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

[98] Uwe Schmidt and Stefan Roth. Shrinkage fields for effective image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2774–2781, 2014.

[99] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[100] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.

[101] Yang Song, Chenlin Meng, and Stefano Ermon. Mintnet: Building invertible neural networks with masked convolutions. In *Advances in Neural Information Processing Systems*, pages 11002–11012, 2019.

[102] James Joseph Sylvester. On the relation between the minor determinants of linearly equivalent quadratic functions. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1(4):295–305, 1851.

[103] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[104] Dustin Tran, Keyon Vafa, Kumar Krishna Agrawal, Laurent Dinh, and Ben Poole. Discrete flows: Invertible generative models of discrete data. *arXiv preprint arXiv:1905.10347*, 2019.

[105] Brian L Trippe and Richard E Turner. Conditional density estimation with Bayesian normalising flows. *arXiv preprint arXiv:1802.04908*, 2018.

[106] Stavros Tsogkas, Iasonas Kokkinos, George Papandreou, and Andrea Vedaldi. Deep learning for semantic part segmentation with high-level guidance. *arXiv preprint arXiv:1505.02438*, 2015.

[107] Lifu Tu and Kevin Gimpel. Learning approximate inference networks for structured prediction. *arXiv preprint arXiv:1803.03376*, 2018.

[108] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1:1–305, 2008.

[109] Shenlong Wang, Sanja Fidler, and Raquel Urtasun. Proximal deep structured models. In *Advances in Neural Information Processing Systems*, pages 865–873, 2016.

[110] Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019.

[111] Max A Woodbury. *Inverting modified matrices*. Statistical Research Group, 1950.

[112] Hao Wu, Jonas Köhler, and Frank Noé. Stochastic normalizing flows. *arXiv preprint arXiv:2002.06707*, 2020.

[113] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29:82–90, 2016.

[114] Rundi Wu, Xuelin Chen, Yixin Zhuang, and Baoquan Chen. Multimodal shape completion via conditional generative adversarial networks. *arXiv preprint arXiv:2003.07717*, 2020.

[115] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[116] Jun Xu, Lei Zhang, David Zhang, and Xiangchu Feng. Multi-channel weighted nuclear norm minimization for real color image denoising. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1096–1104, 2017.

[117] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4541–4550, 2019.

[118] Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 5485–5493, 2017.

[119] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[120] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative

adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[121] Mohsen Zand, Ali Etemad, and Michael Greenspan. Flow-based autoregressive structured prediction of human motion. *arXiv preprint arXiv:2104.04391*, 2021.

[122] Chengxi Zang and Fei Wang. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 617–626, 2020.

[123] Dingwen Zhang, Junwei Han, Gong Cheng, and Ming-Hsuan Yang. Weakly supervised object localization and detection: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[124] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915, 2017.

[125] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[126] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

[127] Dengyong Zhou, Qiang Liu, John C Platt, Christopher Meek, and Nihar B

Shah. Regularized minimax conditional entropy for crowdsourcing. *arXiv preprint arXiv:1503.07240*, 2015.

[128] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National science review*, 5(1):44–53, 2018.

[129] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of International Conference on Computer Vision*, pages 2223–2232, 2017.

[130] Zachary M Ziegler and Alexander M Rush. Latent normalizing flows for discrete sequences. *arXiv preprint arXiv:1901.10548*, 2019.

Appendix A

Experiment Details for Chapter 3

A.1 Experiments of Quantitative Evaluation

In the experiments of qualitative evalution, we compare Woodbury transformations with 3 permutation layer baselines, i.e., 1x1 convolution, emerging convolution, and periodic coupling, and 2 coupling layer baselines, i.e., neural spline coupling, and MaCow. For all generalized permutation methods, we use affine coupling, which is composed of 3 convolutional layers, and the 2 latent layers have 512 channels. For the neural spline coupling, we set the number of spline bins to 4. The spline parameters are generated by a neural network, which is also composed of convolutional layers. For 32×32 images, we set the number of channels to 256, and for 64×64 images, we set it to 224. Ma et al. [69] used steps containing a MaCow unit, i.e., 4 autoregressive convolution coupling layers, and a full Glow step. For fair comparison, we directly use the MaCow unit to replace the affine coupling. For 32×32 images, we set the convolution channel to 384, and for 64×64 images, we set it to 296.

We run each method to fixed number of iterations and test it every $10,000$ iterations. The bpds reported in our main paper are the best bpds obtained by each method. The bpds are single-run results. This is because each run of the experiment requires 3 to 5 days, and running each model multiple times is a major cost. We found in our experiments that for the same model and parameter settings, the bpds' standard deviation of multiple runs are very small, i.e., around 0.003, so single run results are sufficient for comparing bpd.

A.2 Hyper-parameter Settings

We use Adam [49] to tune the learning rates, with $\alpha = 0.001$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We use uniform dequantization. The sizes of models we use, and mini-batch sizes for training in our experiments are listed in Table A.1.

Table A.1: Model sizes and mini-batch sizes.

Dataset	Mini-batch size	Levels(L)	Steps(K)	Coupling channels
CIFAR-10 32x32	64	3	8	512
ImageNet 32x32	64	3	8	512
ImageNet 64x64	32	4	16	512
LSUN Church 96x96	16	5	16	256
CelebA-HQ 64x64	8	4	16	512
CelebA-HQ 128x128	4	5	24	256
CelebA-HQ 256x256	4	6	16	256

A.3 Latent Dimension Settings

In all our experiments, we set the latent dimensions of Woodbury transformations, and ME-Woodbury transformations as in Table A.2.

A.3. Latent Dimension Settings

Table A.2: Latent dimensions of Woodbury transformations and ME-Woodbury transformations. The numbers in the brackets represent the latent dimension used in that level. For example, the $d_c : \{8, 8, 16\}$, represents that the settings of d_c at the three levels are 8, 8, and 16.

Dataset	Woodbury	ME-Woodbury
CIFAR-10 32x32	$d_c : \{8, 8, 16\}$ $d_s : \{16, 16, 8\}$	$d_c : \{8, 8, 16\}$ $d_h : \{16, 16, 8\}$ $d_w : \{16, 16, 8\}$
ImageNet 32x32	$d_c : \{8, 8, 16\}$ $d_s : \{16, 16, 8\}$	$d_c : \{8, 8, 16\}$ $d_h : \{16, 16, 8\}$ $d_w : \{16, 16, 8\}$
ImageNet 64x64	$d_c : \{8, 8, 16, 16\}$ $d_s : \{16, 16, 8, 8\}$	$d_c : \{8, 8, 16, 16\}$ $d_h : \{16, 16, 8, 8\}$ $d_w : \{16, 16, 8, 8\}$
LSUN Church 96x96	$d_c : \{8, 8, 16, 16, 16\}$ $d_s : \{16, 16, 16, 8, 8\}$	—
CelebA-HQ 64x64	$d_c : \{8, 8, 16, 16\}$ $d_s : \{16, 16, 8, 8\}$	—
CelebA-HQ 128x128	$d_c : \{8, 8, 16, 16, 16\}$ $d_s : \{16, 16, 16, 8, 8\}$	—
CelebA-HQ 256x256	$d_c : \{8, 8, 16, 16, 16, 16\}$ $d_s : \{16, 16, 16, 16, 8, 8\}$	—

A.4 Sample Quality Comparisons

We compare the samples generated by Woodbury-Glow and Glow models trained on the CelebA-HQ dataset. We follow Kingma and Dhariwal [51] and randomly hold out 3,000 images as a test set. We use 5-bits images. We use 64×64, 128×128, 256×256 images. Due to the our limited computing resources, we use relatively small models. The model sizes and other settings are listed in Table A.1 and Table A.2. We generate samples from the models during different phases of training and display them in Figure A.1, and Figure A.2 (The results of 64×64 images are shown in the main paper). For the 128×128 images, both Glow and Woodbury-Glow generate distorted images at iteration 100,000, but Woodbury-Glow seems to improve in later stages, stabilizing the shapes of faces and structure of facial features. Glow, continues generating faces with distorted overall shapes as training continues. For the 256×256 images, neither model ever trains sufficiently to generate highly realistic faces, but Woodbury-Glow makes significantly more progress in these 300,000 iterations than Glow. Glow's samples at 300,000 are still mostly random swirls with an occasional recognizable face, while almost all of Woodbury-Glow's samples look like faces, though distorted. Due to limits on our computational resources, we stopped the higher resolution experiments at 300,000 iterations (rather than running to 600,000 iterations as we did for the 64×64 experiments in the main paper). With a larger model and longer training time, it seems Woodbury-Glow would reach higher sample quality much faster than Glow.

The likelihoods of test set under the trained model are listed in Table 3. For the 64×64 and 128×128 images, Woodbury-Glow scores higher likelihood than Glow. For the 256×256 images, their likelihoods are almost identical, and are better than the score reported in [51]. This may be due to three possible reasons: (1) We use affine coupling rather than additive coupling, which is a non-volume preserving layer and may improve the likelihoods;

(2) Since the test set is randomly collected, it is different from the one used in [51]; And (3) The model used in [51] is very large, so it may be somewhat over-fitting. Surprisingly, the clear difference in sample quality is not reflected by the likelihoods. This discrepancy may be because we use 5-bit images, and the images are all faces, so the dataset is less complicated than other datasets such as ImageNet. Moreover, even though Glow cannot generate reasonable 256×256 samples, the colors of these samples already match the colors of real images well, so these strange samples may non-intuitively be equivalently likely as the face-like samples from Woodbury-Glow.

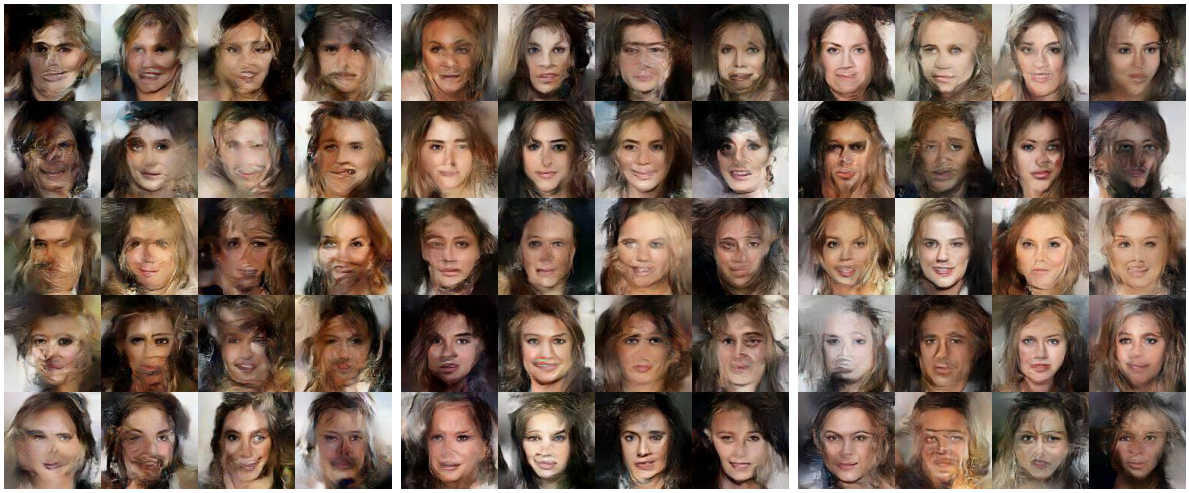
Table A.3: Bit per-dimension results on CelebA-HQ

Size of images	Glow	Woodbury-Glow
64×64	1.27	**1.23**
128×128	1.09	**1.04**
256×256	**0.93**	**0.93**

Woodbury-Glow
Iteration 100,000

Woodbury-Glow
Iteration 200,000

Woodbury-Glow
Iteration 300,000



Glow
Iteration 100,000

Glow
Iteration 200,000

Glow
Iteration 300,000

Figure A.1: Random samples of 128×128 images drawn with temperature 0.7 from a model trained on CelebA data.

A.4. Sample Quality Comparisons

Woodbury-Glow Iteration 150,000 | Woodbury-Glow Iteration 220,000 | Woodbury-Glow Iteration 300,000

Glow Iteration 150,000 | Glow Iteration 220,000 | Glow Iteration 300,000

Figure A.2: Random samples of 256×256 images drawn with temperature 0.7 from a model trained on CelebA data.

A.5 Additional Samples

In this section, we include additional samples from Woodbury-Glow models trained on our various datasets. These samples complement our quantitative analysis. We train our models on CIFAR-10 [57], ImageNet [95], the LSUN church dataset [119], and the CelebA-HQ dataset [48]. Specifically, for ImageNet, we use 32×32 and 64×64 images. For the LSUN dataset, we use the same approach as Kingma and Dhariwal [51] to resize the images to be 96×96. For the CelebA-HQ dataset, we use 64×64, 128×128, and 256×256 images. For LSUN and CelebA-HQ datasets, we use 5-bit images. The parameter settings of our models are in Table A.1 and Table A.2. The samples are in Figures A.3, A.4, A.5, A.6, A.7, A.8, and A.9.

A.5. Additional Samples

Figure A.3: CIFAR-10 32×32 Woodbury-Glow samples.

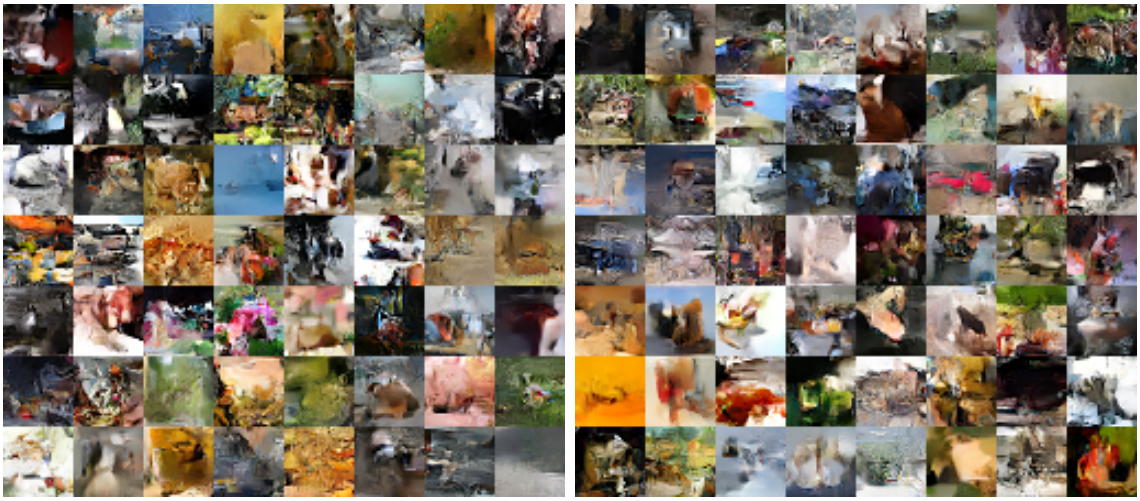


Figure A.4: ImageNet 32×32 Woodbury-Glow samples.

Figure A.5: ImageNet 64×64 Woodbury-Glow samples.

Figure A.6: LSUN church 96×96 Woodbury-Glow samples (temperature 0.875).

A.5. Additional Samples

Figure A.7: CelebA-HQ 64×64 Woodbury-Glow samples (temperature 0.7).

Figure A.8: CelebA-HQ 128×128 Woodbury-Glow samples (temperature 0.5).

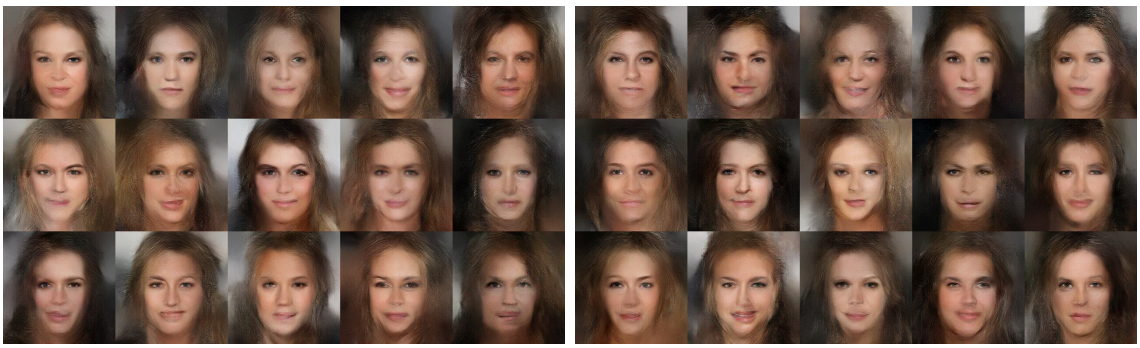


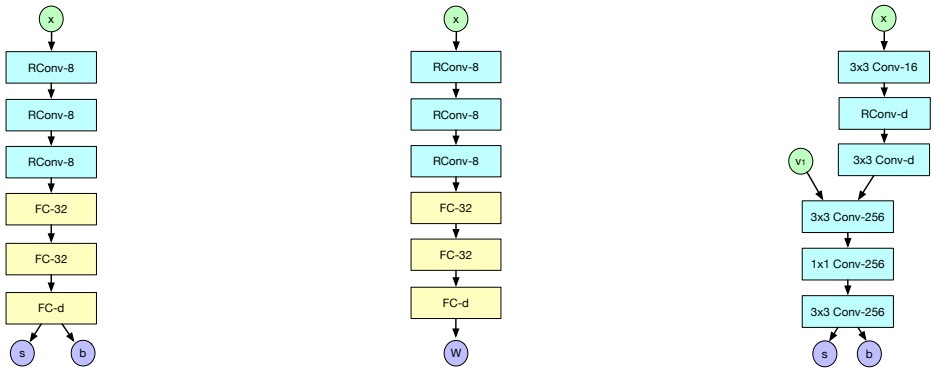
Figure A.9: Selected CelebA-HQ 256×256 Woodbury-Glow samples (temperature 0.5).

Appendix B

Experiment Details for Chapter 4

B.1 Network Architectures

Figure B.1 illustrates the architectures of conditioning networks that we use in our experiments. For each layer except for the last layer, we use ReLU to activate the output. As in Glow, we use zero initialization for each layer. That is, we initialize the weights of each layer to be zero.

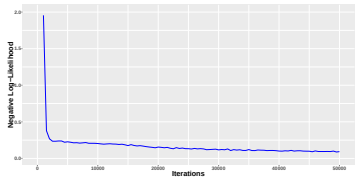


(a) Conditional Actnorm (b) Conditional 1x1 Convolution (c) Conditional Affine

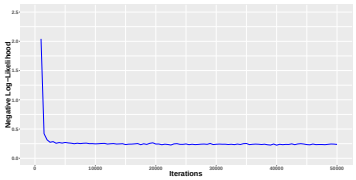
Figure B.1: The networks we use to generate weights. The component "3x3 Conv-256" is a convolutional layer, the kernel size is 3×3, and the number of channels is 256. The component "FC-32" is a fully connected layer, and its width is 32. The parameter d depends on other variable sizes. In the conditional affine layer, d equals the number of channels of v_1. In the conditional actnorm layer, $d = 2c$, where c is the size of the scale. In the conditional 1x1 convolutional layer, W is a $c \times c$ matrix, so $d = c^2$. The "RConv" component is the convolutional layer for downscaling the input.

B.2 Conditional Likelihoods

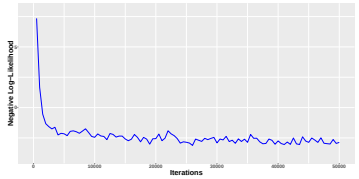
To the best of our knowledge, c-Glow is the first deep structured prediction model whose exact likelihood is tractable. Figure B.2 plots the evolution of minibatch negative log likelihoods during training. Since c-Glow learns a continuous density, the negative log likelihoods can become negative as the model better fits the data distribution.



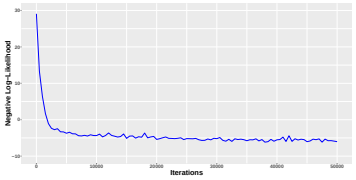
(a) Binary Segmentation



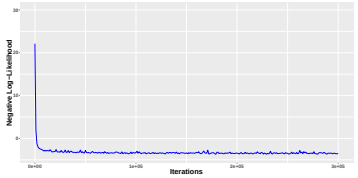
(b) Multi-class Segmentation



(c) Color Image Denoising



(d) Depth Refinement



(e) Image Inpainting

Figure B.2: Evolution of likelihoods.

B.3 Conditional Samples and Predictions

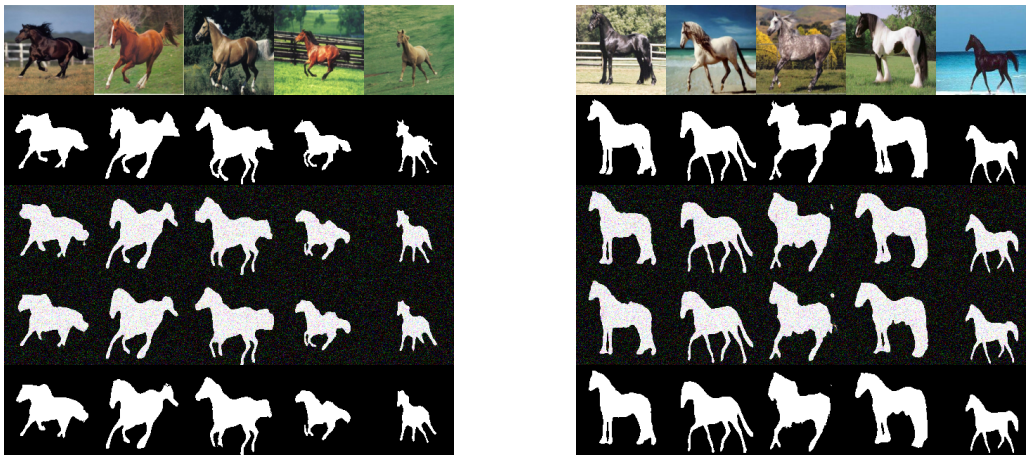
One important advantage of our model is the ability to easily generate high quality conditional samples. In this section, we show some conditional samples as well as prediction results in the following figures. Specifically, Figure B.3 shows the binary segmentation results on the Horses dataset, Figure B.4 shows the multi-class segmentation results on the LFW dataset, Figure B.5 shows the denoised images on the BSDS dataset, Figure B.6 shows the refined depth images on the seven scenes dataset, and Figure B.7 shows the impainted images on the CelebA dataset. For each image except for Figure B.5, we show conditional samples in the third and fourth rows. For the experiments of color image denoising, since the conditional samples are just noise added to images, we omit them in Figure B.5.

For the experiments on semantic segmentation, i.e., Figure B.3 and Figure B.4, the conditional samples are continuous. However as shown in the figures, the generated continuous values are already close to integral. The conditional samples can reflect the divergence in predictions. For example, in the samples of horses, i.e., Figure B.3, some sampled horses have different shapes of heads or tails, even though they are conditioned on the same input image. Similar phenomena can also be seen in Figure B.4.

For image denoising and depth refinement, the outputs are continuous. As discussed in Section 5, the denoised images are not as smooth as those images obtained by feed-forward networks. However, in Figure B.5 and Figure B.6, we can see that c-Glow can remove a large amount of noise from the noisy images and perform reasonably well.

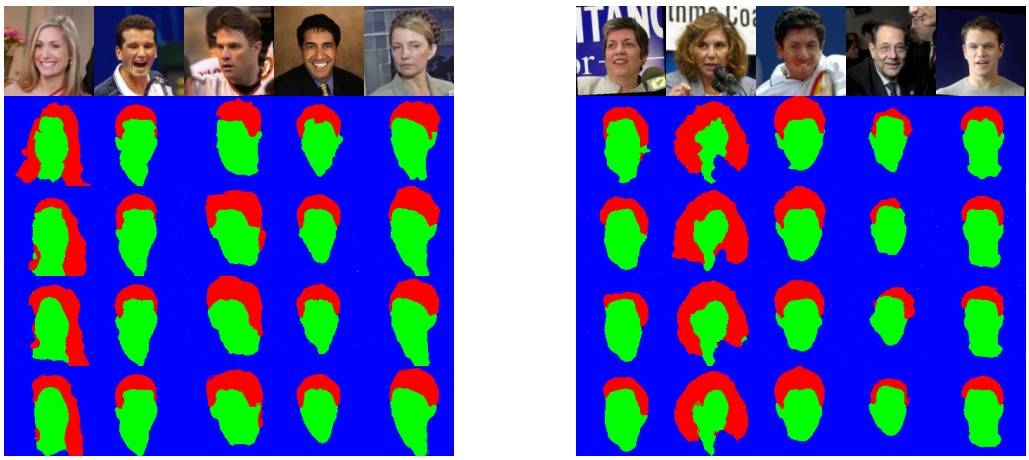
For image inpainting, the images inpainted by c-Glow are slightly inconsistent with the surrounding pixels in color, but they are nearly the same. The inpainted images can capture the shapes of features well, even though for faces with sunglasses, c-Glow can also recover the shape and color of the sunglasses. This is why c-Glow can outperform DCGANi. The

conditional samples of inpainted images can also reflect the diversity of the model predictions. For example, for some samples of faces, parts such as the mouth and nose are the same, but the eyes stare in different directions.



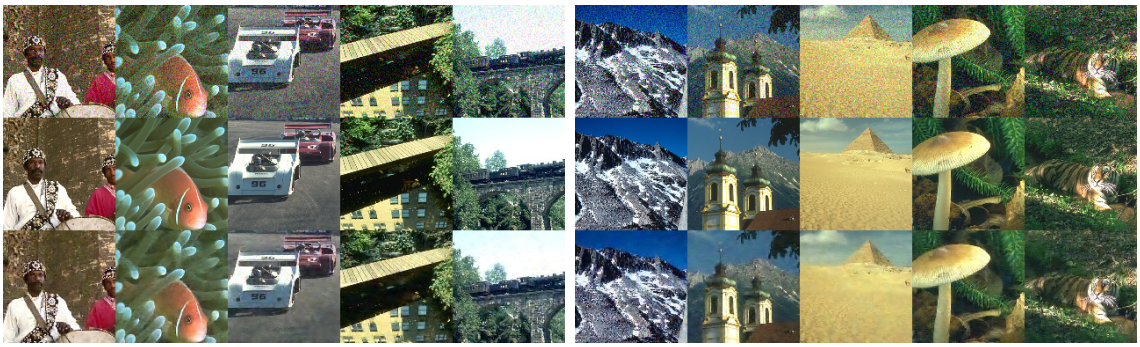
(a) (b)

Figure B.3: Conditional samples and predictions on the Horses dataset. The first two rows are input images and ground truth labels, the third and fourth rows are conditional samples, and the last row is predicted labels.



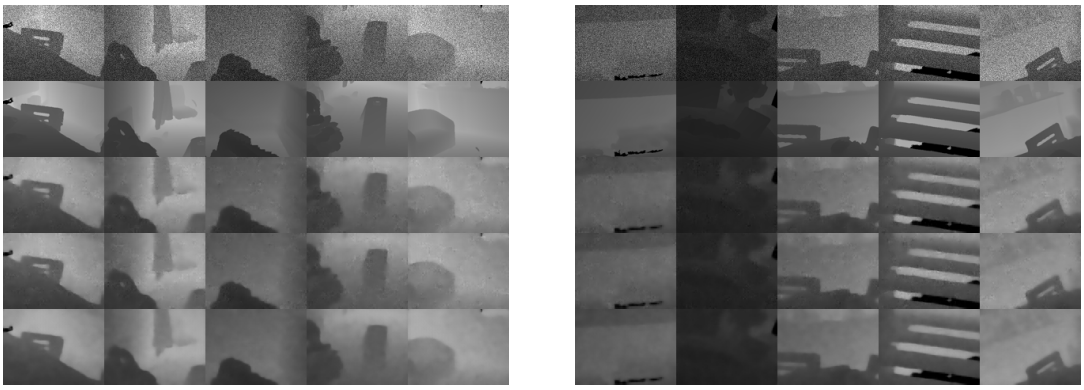
(a) (b)

Figure B.4: Conditional samples and predictions on the LFW dataset. The first two rows are input images and ground truth labels, the third and fourth rows are conditional samples, and the last row is predicted labels.



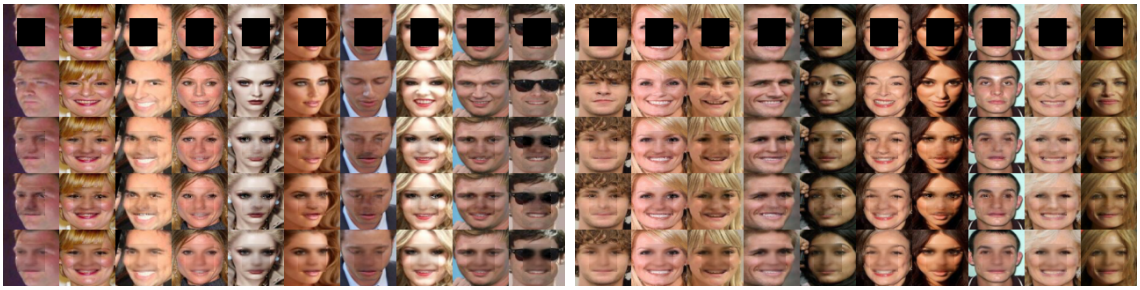
(a) (b)

Figure B.5: Conditional samples and predictions on the BSDS dataset. From top to bottom: the noisy images, the clear images, and the denoised images.



(a) (b)

Figure B.6: Conditional samples and predictions on the seven scenes dataset. The first two rows are input images and ground truth labels, the third and fourth rows are conditional samples, and the last row is predicted labels.



(a) (b)

Figure B.7: Inpainting results on the CelebA dataset. The first row is the corrupted input. The second row is the ground truth labels. The third and fourth rows are conditional samples, and the last row is the inpainting results. We use sample average as the final prediction.

Appendix C

Experiment Details for Chapter 5

C.1 Proof of Theorem 1

The proof of Theorem 5.1 is similar to the proof of dequantization [37, 103]. The complete proof is as follows.

Proof.

$$\begin{aligned}
\mathbb{E}_{p_{data}(\mathbf{x})}\mathbb{E}_{\mathbf{y}\sim U(\Omega^*)}\left[\log p(\mathbf{y}|\mathbf{x},\phi)\right] &= \sum_{\mathbf{x}} p_{data}(\mathbf{x}) \int_{\mathbf{y}\in\Omega^*} \frac{1}{|\Omega^*|} \log p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \\
&\leq M \sum_{\mathbf{x}} p_{data}(\mathbf{x}) \log \int_{\mathbf{y}\in\Omega^*} p(\mathbf{y}|\mathbf{x}) d\mathbf{y} \\
&= M \sum_{\mathbf{x},\hat{\mathbf{y}}} p_{data}(\mathbf{x},\hat{\mathbf{y}}) \log q(\hat{\mathbf{y}}|\mathbf{x}) \\
&= M\mathbb{E}_{p_{data}(\mathbf{x},\hat{\mathbf{y}})}\left[\log q(\hat{\mathbf{y}}|\mathbf{x})\right] \qquad \text{(C.1)}
\end{aligned}$$

In the second row, we use the properties that $\frac{1}{|\Omega^*|} \leq M$, and the Jensen's inequality. In the third row, we use the assumption that $p_{\text{data}}(\mathbf{x}) = p_{\text{data}}(\mathbf{x},\widehat{\mathbf{y}})$, and the relationship between $p(\mathbf{y}|\mathbf{x})$ and $q(\hat{\mathbf{y}}|\mathbf{x})$.

□

C.2 Label Learning Flow for Unpaired point Cloud Completion

In this section, we provide complete derivations of LLF for unpaired point cloud completion. The conditional likelihood $\log p(\mathbf{y}|\mathbf{x}_p)$ is an exchangeable distribution. We use De Finetti's representation theorem and variational inference to derive a tractable lower bound for it.

$$\begin{aligned}
\log p(\mathbf{y}|\mathbf{x}_p) &= \int p(\mathbf{y}, \mathbf{u}|\mathbf{x}_p) d\mathbf{u} \\
&= \int p(\mathbf{y}|\mathbf{u}, \mathbf{x}_p) p(\mathbf{u}) d\mathbf{u} \\
&\geq \mathbb{E}_{q(\mathbf{u}|\mathbf{x}_p)} \left[\log p(\mathbf{y}|\mathbf{u}, \mathbf{x}_p)\right] - \mathrm{KL}(q(\mathbf{u}|\mathbf{x}_p)||p(\mathbf{u})) \\
&\geq \mathbb{E}_{q(\mathbf{u}|\mathbf{x}_p)} \left[\sum_{i=1}^{T_c} \log p(\mathbf{y}_i|\mathbf{u}, \mathbf{x}_p)\right] - \mathrm{KL}(q(\mathbf{u}|\mathbf{x}_p)||p(\mathbf{u})), \qquad \text{(C.2)}
\end{aligned}$$

where in the third row, we use Jensen's inequality to compute the lower bound, and in the last row, we use De Finetti's theorem to factorize $p(\mathbf{y}|\mathbf{u}, \mathbf{x}_p)$ to the distributions of points.

The least square GAN discriminator and the Hausdorff distance for generated complete point clouds can be treated as two equality constraints

$$\begin{aligned}
D(\mathbf{y}) &= 1 \\
d_H(\mathbf{y}, \mathbf{x}_p) &= 0.
\end{aligned}$$

Note that the $d_H()$ is non-negative. Convert these two constraints to penalty functions, we

have

$$\max_{\phi} \quad \mathbb{E}_{q(\mathbf{u}|\mathbf{x}_p)}\left[\sum_{t=1}^{T_c}\log p_Z(\mathbf{z_t}) - \sum_{i=1}^{K}\log\left|\det\left(\frac{\partial \mathbf{g}_{\mathbf{u},\mathbf{x}_p,\phi_i}}{\partial \mathbf{r}_{t,i}}\right)\right|\right] - \mathrm{KL}(q(\mathbf{u}|\mathbf{x}_p)||p(\mathbf{u}))$$
$$-\mathbb{E}_{q(\mathbf{u}|\mathbf{x}_p)}\left[\lambda_1(D(\mathbf{g}_{\mathbf{u},\mathbf{x}_p,\phi}(\mathbf{z}))-1)^2 + \lambda_2 d^{HL}(\mathbf{g}_{\mathbf{u},\mathbf{x}_p,\phi}(z),\mathbf{x}_p)\right]. \tag{C.3}$$

C.3 Experiment Details

In this section, we provide more details on our experiments to help readers reproduce our results.

C.3.1 Model Architecture

For experiments of weakly supervised classification, and unpaired point cloud completion, we use normalizing flows with only conditional affine coupling layers [54]. Each layer is defined as

$$\begin{aligned}
&\mathbf{y}_a, \mathbf{y}_b = \mathrm{split}(\mathbf{y}) \\
&\mathbf{s} = \mathbf{m}_s(\mathbf{w}_y(\mathbf{y}_a) \odot \mathbf{w}_x(\mathbf{x}) + \mathbf{w}_b(\mathbf{x})) \\
&\mathbf{b} = \mathbf{m}_b(\mathbf{c}_y(\mathbf{y}_a) \odot \mathbf{c}_x(\mathbf{x}) + \mathbf{c}_b(\mathbf{x})) \\
&\mathbf{z}_b = \mathbf{s} \odot \mathbf{y}_b + \mathbf{b} \\
&\mathbf{z} = \mathrm{concat}(\mathbf{y}_a, \mathbf{z}_b),
\end{aligned} \tag{C.4}$$

where $\mathbf{m}, \mathbf{w}, \mathbf{c}$ are all small neural networks.

For LLF, we only need the inverse flow, i.e., $\mathbf{g}_{\mathbf{x},\phi}$, for training and prediction, so in our experiments, we actually define $\mathbf{g}_{\mathbf{x},\phi}$ as the forward transformation, and let $\mathbf{y} = \mathbf{s} \odot \mathbf{z} + \mathbf{b}$.

We do this because multiplication and addition are more stable than division and subtraction.

Weakly Supervised Classification

In this problem, we use a flow with 8 flow steps, and each step has 2 conditional affine coupling layers. These two layers will transform different dimensions. Each $\mathbf{w}$ and $\mathbf{c}$ are small MLPs with two linear layers. Each $\mathbf{m}$ has one linear layer. The hidden dimension of linear layers is fixed to 64.

Weakly Supervised Regression

In this problem, since the label y is a scalar, we use conditional affine transformation introduced in Section 5.4, as a flow layer. A flow has 8 flow layers. The $\mathbf{s}$ and $\mathbf{b}$ in a flow layer are three layer MLPs. The hidden dimension of linear layers is 64.

Unpaired Point Cloud Completion

The model architecture used LLF used for this problem is illustrated in Figure C.1. We use the same architecture as DPF [54] for point flow. Specifically, the flow has 8 flow steps, and each step has 3 conditional affine coupling layers, i.e., Equation C.4. Slightly different from the original DPF, the conditioning networks $\mathbf{w}_x$, $\mathbf{c}_x$, $\mathbf{w}_b$, and $\mathbf{c}_b$ will take the latent variable $\mathbf{u}$ and the features of partial point cloud $\mathbf{x}_p$ as input. The $\mathbf{w}$s and $\mathbf{c}$s are MLPs with two linear layers, whose hidden dimension is 64. The $\mathbf{m}$s are one layer MLPs.

We use a PointNet [87] to extract features from partial point cloud $\mathbf{x}_p$. Following [54], the hidden dimensions of this PointNet is set as $64 - 128 - 256 - 512$. Given the features of $\mathbf{x}_p$, the encoder E then uses the reparameterization trick [50] to generate latent variable $\mathbf{u}$. The encoder has two linear layers, whose hidden dimension is 512.

The GAN discriminator uses another PointNet to extract features from (generated) complete point clouds. We follow [114] and set the hidden dimensions of this PointNet as $64 - 128 - 128 - 256 - 128$. The discriminator D is a three layer MLP, whose hidden dimensions are $128 - 256 - 512$.

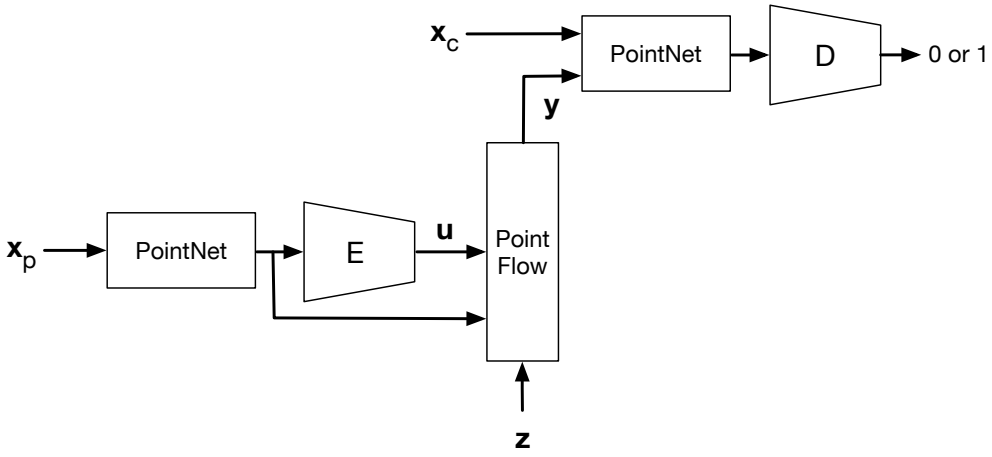


Figure C.1: Model architecture of LLF for unpaired point cloud completion. The E represents the encoder, and the D represents the GAN discriminator.

C.3.2 Data, Training, and Evaluation Details

Weakly Supervised Classification

For experiments on tabular and image datasets, we use the same approach as [3, 4] to split each dataset to training, simulation, and test sets. We use the data and labels in simulation sets to create weak signals, and estimated bounds. We train models on training sets and test model on test sets. We assume that the models do not have access to any labels. The labels in simulation sets are only used to generated weak signals and estimate bounds. We follow [4] and choose 3 features to create weak signals. We train a logistic regression with each feature on the simulation set, and use the label probabilities predicted by this logistic regression as weak signals. We compute the error of this trained logistic regression on simulation set as

estimated error bound.

For experiments on real text datasets, we use the same keyword-based method as [3] to create weak supervision. Specifically, we choose key words that can weakly indicate positive and negative sentiments. Documents containing positive words will be labeled as positive, and vice versa. For two-stage methods, we follow [3] and use a two layer MLP as the classifier, whose latent dimension is 512.

We list some main features of these datasets in Table C.1. We refer to their original papers for more details. For those datasets without official splits, we randomly split them with a ratio of $4:3:3$.

Table C.1: Summary of datasets used in weakly supervised classification experiments. The "—" indicates this dataset does not have a official split

Dataset	Size	Train Size	Test Size	No. features	No. weak signals
Fashion MNIST (DvK)	$14,000$	$12,000$	$2,000$	784	3
Fashion MNIST (SvA)	$14,000$	$12,000$	$2,000$	784	3
Fashion MNIST (CvB)	$14,000$	$12,000$	$2,000$	784	3
Breast Cancer	569	—	—	30	3
OBS Network	795	—	—	21	3
Cardiotocography	963	—	—	21	3
Clave Direction	$8,606$	—	—	16	3
Credit Card	$1,000$	—	—	24	3
Statlog Satellite	$3,041$	—	—	36	3
Phishing Websites	$11,055$	—	—	30	3
Wine Quality	$4,974$	—	—	11	3
IMDB	$49,574$	$29,182$	$20,392$	300	10
SST	$5,819$	$3,998$	$1,821$	300	14
YELP	$55,370$	$45,370$	$10,000$	300	14

Weakly Supervised Regression

We use three datasets from the UCI repository. For each dataset, we randomly split it to training, simulation, and test sets with a ratio of $4:3:3$. We use the simulation set to create weak signals and estimated label values. We choose 5 features to create weak signals for each dataset. The detailed introduction of these datasets are as follows. Table C.2 summarize the statistical results of them.

Air Quality. In this task, we predict the absolute humidity in air, based on other air quality features such as hourly averaged temperature, hourly averaged NO_2 concentration etc. The raw dataset has $9,358$ instances. We remove those instances with Nan values, resulting in a dataset with $8,991$ instances. We use hourly averaged concentration CO, hourly averaged Benzene concentration, hourly averaged NO_x concentration, tungsten oxide hourly averaged sensor response, and relative humidity as features for creating weak signals.

Temperature Forecast. In this task, we predict the next day maximum air temperature based on current day information. The raw dataset has $7,750$ instances, and we remove those instances with Nan values, resulting in $7,588$ instances. We use present max temperature, forecasting next day wind speed, forecasting next day cloud cover, forecasting next day precipitation, solar radiation as features for creating weak signals.

Bike Sharing. In this task, we predict the count of total rental bikes given weather and date information. The raw dataset has $17,389$ instances, and we remove those instances with Nan values, resulting in $17,379$ instances. We use season, hour, if is working day, normalized feeling temperature, and wind speed as features for creating weak signals.

Table C.2: Summary of datasets used in weakly supervised regression experiments

Dataset	Size	No. features	No. weak signals
Air Quality	8,991	12	5
Temperature Forecast	7,588	24	5
Bike Sharing	17,379	12	5

Unpaired Point Cloud Completion

Datasets. We use the same way as [114] to process PartNet [77]. PartNet provides point-wise semantic labels for point clouds. The original point clouds are used as complete point clouds. To generate partial point clouds, we randomly removed parts from complete point clouds, based on the semantic labels. We use Chair, Table, and Lamp categories. The summary of these three subsets are in Table C.3.

Table C.3: Summary of datasets used unpaird point cloud completion experiments

Dataset	Train Size	Valid Size	Test Size
Chair	4,489	617	1,217
Table	5,707	843	1,668
Lamp	1,545	234	416

Metrics. Let $\mathcal{X}_c$ be the set of referred complete point clouds, and $\mathcal{X}_p$ be the set of input partial point clouds. For each partial point cloud $\mathbf{x}_i^{(p)}$, we generate M complete point cloud samples $\mathbf{y}_i^{(1)}, ..., \mathbf{y}_i^{(M)}$. All these samples form a new set of complete point clouds $\mathcal{Y}$. In our experiments, we follow [114] and set $M = 10$.

The MMD [2] is defined as

$$\text{MMD} = \frac{1}{|\mathcal{X}_c|} \sum_{\mathbf{x}_i \in \mathcal{X}_c} d_C(\mathbf{x}_i, \text{NN}(\mathbf{x}_i)), \tag{C.5}$$

where $\mathrm{NN}(\mathbf{x})$ is the nearest neighbor of $\mathbf{x}$ in $\mathcal{Y}$. The d_C represents Chamfer distance. MMD computes the distance between the set of generated samples and the set of target complete shapes, so it measures the quality of generated.

The TMD is defined as

$$\mathrm{TMD} = \frac{1}{|\mathcal{X}_p|} \sum_{i=1}^{|\mathcal{X}_p|} \left(\frac{2}{M-1} \sum_{j=1}^{M} \sum_{k=j+1}^{M} d_C(\mathbf{y}_i^{(j)}, \mathbf{y}_i^{(k)}) \right). \tag{C.6}$$

TMD measures the difference of generated samples given an input partial point cloud, so it measures the diversity of samples.

The UHD is defined as

$$\mathrm{UHD} = \frac{1}{|\mathcal{X}_p|} \sum_{i=1}^{|\mathcal{X}_p|} \left(\frac{1}{M} \sum_{j=1}^{M} d_H(\mathbf{x}_i, \mathbf{y}_i^{(j)}) \right), \tag{C.7}$$

where d_H represents the unidirectional Hausdorff distance. UHD measures the similarity between generated samples and input partial point clouds, so it measures the fidelity of samples.

Samples. We compare LLF to mm-pc2pc in Figure 5.2, and more samples of LLF in Figure C.2, Figure C.3, and Figure C.4. LLF can generate samples that are as good as mm-pc2pc. Mm-pc2pc has a higher diversity in samples, but some generated samples may be unreasonable.

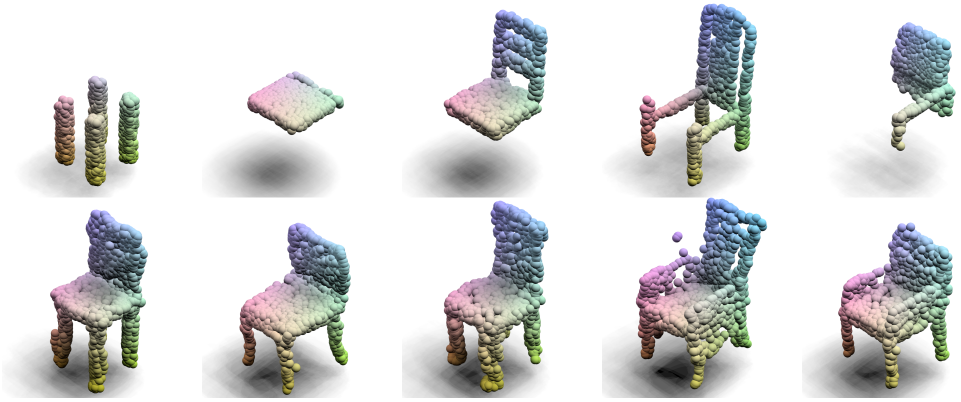


Figure C.2: Random chair samples generated by LLF. The first row is partial point clouds, and the second row is generated complete point clouds.

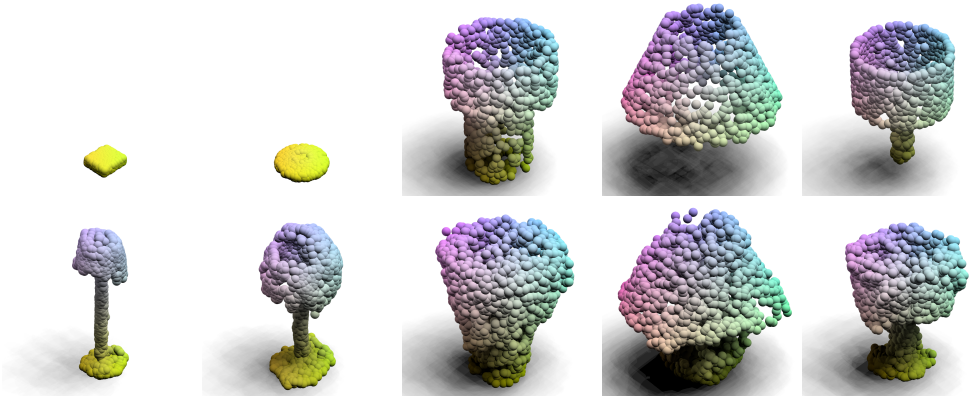


Figure C.3: Random lamp samples generated by LLF.

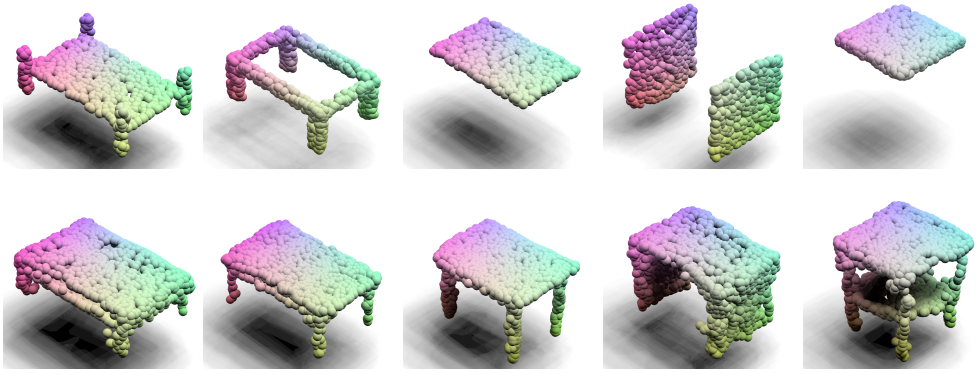


Figure C.4: Random table samples generated by LLF.