Structured Output Learning with Conditional Generative Flows

You Lu
Department of Computer Science
Virginia Tech
Blacksburg, VA
you.lu@vt.edu

Bert Huang
Department of Computer Science
Virginia Tech
Blacksburg, VA
bhuang@vt.edu

Abstract

Traditional structured prediction models try to learn the conditional likelihood, i.e., $p(y|x)$, to capture the relationship between the structured output y and the input features 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. In this paper, we propose conditional Glow (c-Glow), a conditional generative flow for structured output learning. C-Glow benefits from the ability of flow-based models to compute $p(y|x)$ 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 versatile and is applicable to many different structured prediction problems.

1 Introduction

Structured prediction models are widely used in tasks such as image segmentation (Nowozin and Lampert 2011) and sequence labeling (Lafferty, McCallum, and Pereira 2001). In these structured output tasks, the goal is to model a mapping from the input x to the high-dimensional structured output y. In many such problems, it is also important to make diverse predictions to capture the variability of plausible solutions to the structured output problem (Sohn, Lee, and Yan 2015).

Many existing methods for structured output learning use graphical models, such as conditional random fields (CRFs) (Wainwright and Jordan 2008), 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 (Chen et al. 2015; Zheng et al. 2015; Sohn, Lee, and Yan 2015; Wang, Fidler, and Urtasun 2016; Belanger and McCallum 2016; Graber, Meshi, and Schwing

2018) 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 (Koller and Friedman 2009).

In this paper, we develop conditional generative flows (c-Glow) for structured output learning. Our model is a variant of Glow (Kingma and Dhariwal 2018), 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 (Krähenbühl and Koltun 2011)). 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., (Trippe and Turner 2018; Kingma and Dhariwal 2018)), 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.

2 Related Work

There are two main topics of research related to our paper: deep structured prediction and normalizing flows. In this section, we briefly cover some of the most related literature.

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, Fidler, and Urtasun (2016) proposed proximal methods for inference. And Sohn, Lee, and Yan (2015) used variational autoencoders (Kingma and Welling 2013) 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) (Belanger and McCallum 2016) 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 (Belanger, Yang, and McCallum 2017) based on unrolled optimization. Methods to alleviate the cost of SPEN inference include replacing the argmax inference with an inference network (Tu and Gimpel 2018). Inspired by Q-learning, Gygli, Norouzi, and Angelova (2017) used an oracle value function as the objective for energy-based deep networks. Graber, Meshi, and Schwing (2018) generalized SPENs by adding non-linear transformations on top of the score function.

2.2 Normalizing Flows

Normalizing flows are neural networks constructed with fully invertible components. The invertibility of the resulting network provides various mathematical benefits. Normalizing flows have been successfully used to build likelihood-based deep generative models (Dinh, Krueger, and Bengio 2014; Dinh, Sohl-Dickstein, and Bengio 2016; Kingma and Dhariwal 2018) and to improve variational approximation (Rezende and Mohamed 2015; Kingma et al. 2016). Autoregressive flows (Kingma et al. 2016; Papamakarios, Pavlakou, and Murray 2017; Huang et al. 2018; Ziegler and Rush 2019) condition each affine transformation on all previous variables, so that they ensure an invertible transformation and triangular Jacobian matrix. Continuous normalizing flows (Chen et al. 2018; Grathwohl et al. 2018) define the transformation function using ordinary differential equations. While most normalizing flow models define generative models, Trippe and Turner (2018) developed radial flows to model univariate conditional probabilities.

Most related to our approach are flow-based generative models for complex output. Dinh, Krueger, and Bengio (2014) first proposed a flow-based model, NICE, for modeling complex high-dimensional densities. They later proposed Real-NVP (Dinh, Sohl-Dickstein, and Bengio 2016), which improves the expressiveness of NICE by adding more flexible coupling layers. The Glow model (Kingma and Dhariwal 2018) further improved the performance of such approaches by incorporating new invertible layers. Most recently, Flow++ (Ho et al. 2019) improved generative flows with variational dequantization and architecture design, and Hoogeboom, Berg, and Welling (2019) proposed new invertible convolutional layers for flow-based models.

3 Background

In this section, we introduce notation and background knowledge directly related to our work.

3.1 Structured Output Learning

Let x and y be random variables with unknown true distribution $p^*(y|x)$. We collect a dataset $\mathcal{D} = \{(x_1, y_1), ..., (x_N, y_N)\}$, where x_i is the ith input and y_i is the corresponding output. We approximate $p^*(y|x)$ with a model $p(y|x, \theta)$ and minimize the negative log-likelihood

$$\mathcal{L}(\mathcal{D}) = -\frac{1}{N} \sum_{i=1}^{N} \log p(y_i|x_i, \theta).$$

In structured output learning, the label y comes from a complex, high-dimensional output space $\mathcal{Y}$ with dependencies among output dimensions. Many structured output learning approaches use an energy-based model to define a conditional distribution:

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

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

3.2 Conditional Normalizing Flows

A normalizing flow is a composition of invertible functions $f = f_1 \circ f_2 \circ \cdots \circ f_M$, which transforms the target y to a latent code z drawn from a simple distribution. In conditional normalizing flows (Trippe and Turner 2018), we rewrite each function as $f_i = f_{x,\phi_i}$, making it parameterized by both x and its parameter ϕ_i. Thus, with the *change of variables* formula, we can rewrite the conditional likelihood as

$$\log p(y|x, \theta) = \log p_Z(z) + \sum_{i=1}^{M} \log \left| \det \left(\frac{\partial f_{x,\phi_i}}{\partial r_{i-1}} \right) \right|, \quad (1)$$

where $r_i = f_{\phi_i}(r_{i-1})$, $r_0 = x$, and $r_M = z$.

In this paper, we address the structured output problem by using normalizing flows. That is, we directly use the conditional normalizing flows, i.e., Equation 1, to calculate the conditional distribution. Thus, the model can be trained

by locally optimizing the exact likelihood. Note that conditional normalizing flows have been used for conditional density estimation. Trippe and Turner (2018) 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. We therefore will build on recently developed methods for (unconditional) flow-based generative models for high-dimensional data.

3.3 Glow

Glow (Kingma and Dhariwal 2018) is a flow-based generative model that extends other flow-based models: NICE (Dinh, Krueger, and Bengio 2014) and Real-NVP (Dinh, Sohl-Dickstein, and Bengio 2016). Glow's modifications have demonstrated significant improvements in likelihood and sample quality for natural images. The model mainly consists of three components. Let u and v be the input and output of a layer, whose shape is $[h \times w \times c]$, with spatial dimensions (h, w) and channel dimension c. The three components are as follows.

Actnorm layers. Each activation normalization (actnorm) layer performs an affine transformation of activations using two $1 \times c$ parameters, i.e., a scalar s, and a bias b. The transformation can be written as

$$u_{i,j} = s \odot v_{i,j} + b,$$

where $\odot$ is the element-wise product.

Invertible 1×1 convolutional layers. Each invertible 1x1 convolutional layer is a generalization of a permutation operation. Its function format is

$$u_{i,j} = W v_{i,j},$$

where W is a $c \times c$ weight matrix.

Affine layers. As in the NICE and Real-NVP models, Glow also has affine coupling layers to capture the correlations among spatial dimensions. Its transformation is

$$v_1, v_2 = \text{split}(v), \qquad s_2, b_2 = \text{NN}(v_1),$$
$$u_2 = s_2 \odot v_2 + b_2, \quad u = \text{concat}(v_1, u_2),$$

where NN is a neural network, and the split() and concat() functions perform operations along the channel dimension. The s_2 and b_2 vectors have the same size as v_2.

Glow uses a multi-scale architecture (Dinh, Sohl-Dickstein, and Bengio 2016) to combine the layers. This architecture has a "squeeze" layer for shuffling the variables and a "split" layer for reducing the computational cost.

4 Conditional Generative Flows for Structured Output Learning

This section describes our conditional generative flow (c-Glow), a flow-based model for structured prediction.

4.1 Conditional Glow

To modify Glow to be a conditional generative flow, 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

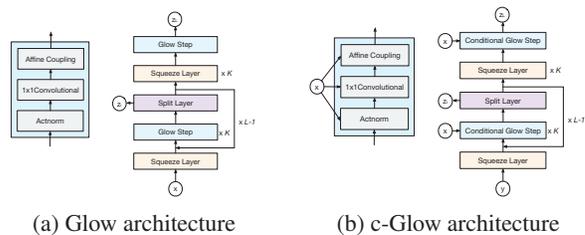(a) Glow architecture   (b) c-Glow architecture

Figure 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.

network, which we refer to as a *conditioning network* (CN), to generate the parameter weights for each layer. The details are as follows.

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

$$s, b = \text{CN}(x), \quad u_{i,j} = s \odot v_{i,j} + b.$$

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:

$$W = \text{CN}(x), \quad u_{i,j} = W v_{i,j}.$$

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

$$v_1, v_2 = \text{split}(v), \qquad x_r = \text{CN}(x),$$
$$s_2, b_2 = \text{NN}(v_1, x_r), \quad u_2 = s_2 \odot v_2 + b_2,$$
$$u = \text{concat}(v_1, u_2).$$

We can still use the multi-scale architecture to combine these conditional components to preserve the efficiency of computation. Figure 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 5.1.

4.2 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. 1, and optimize all c-Glow parameters using gradient methods.

In cases where the output is discrete, we follow (Dinh, Sohl-Dickstein, and Bengio 2016; Kingma and Dhariwal 2018; Ho et al. 2019) 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 backpropagation and gradient methods to optimize the likelihood of this approximate continuous distribution. By expanding the proofs by Theis, Oord, and Bethge (2015) and Ho et al. (2019), we can show that the discrete distribution is lowerbounded by this continuous distribution.

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

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

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

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

$$
\begin{aligned}
&\mathbb{E}_{\tilde{p}_d(x,y)}[\log p(v|x)] \\
=\ & \int_x \sum_y p_d(x, y) dx \int_u \log p(y + u|x) du \\
\leq\ & \int_x \sum_y p_d(x, y) dx \log \int_u p(y + u|x) du \\
=\ & \mathbb{E}_{p_d(x,y)}[\log q(y|x)].
\end{aligned}
$$

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

4.3 Inference

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

$$z \sim p_Z(z), y = g_{x,\phi}(z), \qquad (2)$$

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

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

$$y^* = \arg\max_y p(y|x). \qquad (3)$$

To compute Equation 3, 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:

$$y^* \approx \frac{1}{M} \sum_{i=1}^{M} g_{x,\phi}(z_i). \qquad (4)$$

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 2.

When y is a continuous variable, we can directly get y^* from the above sample-based prediction. When y is discrete, we follow previous literature (Belanger and McCallum 2016; Gygli, Norouzi, and Angelova 2017) to round y^* to discrete values. In our experiments, we find that the predicted y^* values are already near integral values.

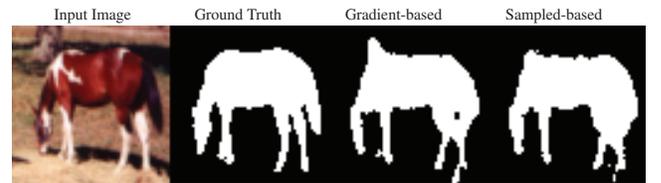| Input Image | Ground Truth | Gradient-based | Sampled-based |

Figure 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.

5 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.

5.1 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 x to a reasonable size. The last three layers are then fully connected layers, which transform the resized x to the scale s and the bias 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 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 x, and we concatenate it with v_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 (Kingma and Ba 2014) 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.

5.2 Binary Segmentation

In this set of experiments, we use the Weizmann Horse Image Database (Borenstein and Ullman 2002), 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 (Gygli, Norouzi, and Angelova 2017), NLStruct (Graber, Meshi, and Schwing 2018), and FCN[1] (Long, Shelhamer, and Darrell 2015). Since the code for DVN and NLStruct is not available online, we reproduce results of DVN and NLStruct by Gygli, Norouzi, and Angelova (2017), and Graber, Meshi, and Schwing (2018). We use mean intersection-over-union (IOU) as the 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 $n_w = 128$.

---

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

Table 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 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. In contrast, 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.

5.3 Multi-class Segmentation

In this set of experiments, we use the Labeled Faces in the Wild (LFW) dataset (Huang, Jain, and Learned-Miller 2007; Kae et al. 2013). 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 (Kae et al. 2013; Gygli, Norouzi, and Angelova 2017), and super-pixel accuracy (SPA) as our metric. Since c-Glow predicts the pixel-wise label, we follow previous papers (Tsogkas et al. 2015; Gygli, Norouzi, and Angelova 2017) 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 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 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 im-

ages, 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 (Long, Shelhamer, and Darrell 2015; Gygli, Norouzi, and Angelova 2017), 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.

5.4 Color Image Denoising

In this section, we conduct color image denoising on the BSDS500 dataset (Arbelaez et al. 2010). We train models on 400 images and test them on the commonly used 68 images (Roth and Black 2009). Following previous work (Schmidt and Roth 2014), 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 (Dabov et al. 2007), DnCNN (Zhang et al. 2017), and McWNNM (Xu et al. 2017). 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 x be the clean images and $\hat{x}$ be the noisy images. To train the model, we follow Zhang et al. (2017) and use $(\hat{x})$ as the input and $\hat{x} - x$ as the output. To denoise the images, we first predict y^* and then compute $(\hat{x} - y^*)$.

Table 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 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 3. One main reason the PSNR of c-Glow is lower than 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 pixelwise checkerboard mask to split the input tensor, which may cause values of neighbor pixels to vary non-smoothly.
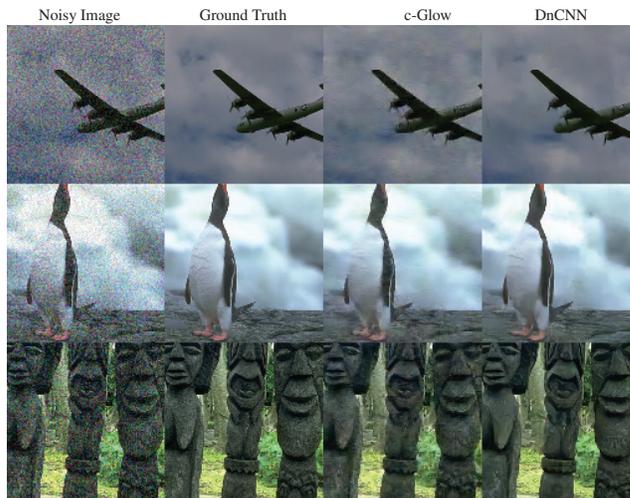

Figure 3: Example qualitative results.

5.5 Denoising for Depth Refinement

In this set of experiments, we use the seven scenes dataset (Newcombe et al. 2011), which contains noisy depth maps of natural scenes. The task is to denoise the depth maps. We use the same method as Wang, Fidler, and Urtasun (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 (Wang, Fidler, and Urtasun 2016), FilterForest (Ryan Fanello et al. 2014), and BM3D (Dabov et al. 2007). 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. 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: Depth refinement scores (PSNR).

c-Glow	ProximalNet	FilterForest	BM3D
36.53	36.31	35.63	35.46

5.6 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 (Liu et al. 2015), 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) (Yeh et al. 2017), which is the state-of-the-art deep model for image inpainting. We use PSNR as our metric.

Table 5: Image inpainting scores (PSNR). "DCGANi-b" represents DCGANi with Poisson blending.

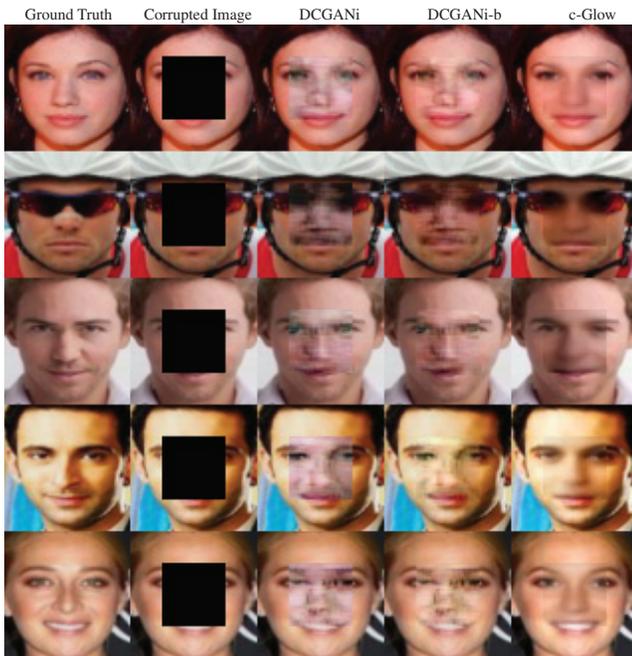| c-Glow	DCGANi-b	DCGANi
24.88	23.65	22.73

Figure 4: Sample results of c-Glow and DCGAN inpainting.

The PSNR scores are in Table 5. Figure 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.

5.7 Discussion

We evaluated c-Glow on five different structured prediction tasks. Two tasks require discrete outputs (binary and multiclass 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.

6 Conclusion

In this paper, 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. In contrast with existing deep structured models, 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.

Acknowledgments

References

Arbelaez, P.; Maire, M.; Fowlkes, C.; and Malik, J. 2010. Contour detection and hierarchical image segmentation. *IEEE Trans. on Pattern Analysis and Mach. Intell.* 33(5):898–916.

Belanger, D., and McCallum, A. 2016. Structured prediction energy networks. In *Intl. Conf. on Machine Learning*, 983–992.

Belanger, D.; Yang, B.; and McCallum, A. 2017. End-to-end learning for structured prediction energy networks. In *Intl. Conf. on Machine Learning*, 429–439.

Borenstein, E., and Ullman, S. 2002. Class-specific, top-down segmentation. In *European Conf. on Comp. Vision*, 109–122.

Chen, L.-C.; Schwing, A.; Yuille, A.; and Urtasun, R. 2015. Learning deep structured models. In *International Conference on Machine Learning*, 1785–1794.

Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 6571–6583.

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

Dinh, L.; Krueger, D.; and Bengio, Y. 2014. NICE: Nonlinear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Dinh, L.; Sohl-Dickstein, J.; and Bengio, S. 2016. Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*.

Graber, C.; Meshi, O.; and Schwing, A. 2018. Deep structured prediction with nonlinear output transformations. In *Advances in Neural Information Processing Systems*, 6320–6331.

Grathwohl, W.; Chen, R. T.; Betterncourt, J.; Sutskever, I.; and Duvenaud, D. 2018. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*.

Gygli, M.; Norouzi, M.; and Angelova, A. 2017. Deep value networks learn to evaluate and iteratively refine structured outputs. In *Proc. of the Intl. Conf. on Machine Learning*, 1341–1351.

Ho, J.; Chen, X.; Srinivas, A.; Duan, Y.; and Abbeel, P. 2019. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*.

Hoogeboom, E.; Berg, R. v. d.; and Welling, M. 2019. Emerging convolutions for generative normalizing flows. *arXiv preprint arXiv:1901.11137*.

Huang, C.-W.; Krueger, D.; Lacoste, A.; and Courville, A. 2018. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779*.

Huang, G. B.; Jain, V.; and Learned-Miller, E. 2007. Unsupervised joint alignment of complex images. In *2007 IEEE 11th International Conference on Computer Vision*, 1–8. IEEE.

Kae, A.; Sohn, K.; Lee, H.; and Learned-Miller, E. 2013. Augmenting CRFs with Boltzmann machine shape priors for image labeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019–2026.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., and Dhariwal, P. 2018. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, 10215–10224.

Kingma, D. P., and Welling, M. 2013. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P.; Salimans, T.; Jozefowicz, R.; Chen, X.; Sutskever, I.; and Welling, M. 2016. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, 4743–4751.

Koller, D., and Friedman, N. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Krähenbühl, P., and Koltun, V. 2011. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Advances in Neural Information Processing Systems*, 109–117.

Lafferty, J.; McCallum, A.; and Pereira, F. C. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Intl. Conf. on Machine Learning*.

Liu, Z.; Luo, P.; Wang, X.; and Tang, X. 2015. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision*.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Computer Vision and Pattern Recognition*, 3431–3440.

Newcombe, R. A.; Izadi, S.; Hilliges, O.; Molyneaux, D.; Kim, D.; Davison, A. J.; Kohli, P.; Shotton, J.; Hodges, S.; and Fitzgibbon, A. W. 2011. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, volume 11, 127–136.

Norouzi, M.; Bengio, S.; Chen, Z.; Jaitly, N.; Schuster, M.; Wu, Y.; and Schuurmans, D. 2016. Reward augmented maximum likelihood for neural structured prediction. In *Advances in Neural Information Processing Systems*, 1723–1731.

Nowozin, S., and Lampert, C. H. 2011. Structured learning and prediction in computer vision. *Foundations and Trends in Computer Graphics and Vision* 6:185–365.

Papamakarios, G.; Pavlakou, T.; and Murray, I. 2017. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, 2338–2347.

Rezende, D. J., and Mohamed, S. 2015. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*.

Roth, S., and Black, M. J. 2009. Fields of experts. *International Journal of Computer Vision* 82(2):205.

Ryan Fanello, S.; Keskin, C.; Kohli, P.; Izadi, S.; Shotton, J.; Criminisi, A.; Pattacini, U.; and Paek, T. 2014. Filter forests for learning data-dependent convolutional kernels. In *Proc. of the IEEE Conf. on Computer Vis. and Pattern Recog.*, 1709–1716.

Schmidt, U., and Roth, S. 2014. Shrinkage fields for effective image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2774–2781.

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, 3483–3491.

Theis, L.; Oord, A. v. d.; and Bethge, M. 2015. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

Trippe, B. L., and Turner, R. E. 2018. Conditional density estimation with Bayesian normalising flows. *arXiv preprint arXiv:1802.04908*.

Tsogkas, S.; Kokkinos, I.; Papandreou, G.; and Vedaldi, A. 2015. Deep learning for semantic part segmentation with high-level guidance. *arXiv preprint arXiv:1505.02438*.

Tu, L., and Gimpel, K. 2018. Learning approximate inference networks for structured prediction. *arXiv preprint arXiv:1803.03376*.

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

Wang, S.; Fidler, S.; and Urtasun, R. 2016. Proximal deep structured models. In *Advances in Neural Information Processing Systems*, 865–873.

Xu, J.; Zhang, L.; Zhang, D.; and Feng, X. 2017. Multi-channel weighted nuclear norm minimization for real color image denoising. In *Proceedings of the IEEE International Conference on Computer Vision*, 1096–1104.

Yeh, R. A.; Chen, C.; Yian Lim, T.; Schwing, A. G.; Hasegawa-Johnson, M.; and Do, M. N. 2017. Semantic image inpainting with deep generative models. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 5485–5493.

Zhang, K.; Zuo, W.; Chen, Y.; Meng, D.; and Zhang, L. 2017. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing* 26(7):3142–3155.

Zheng, S.; Jayasumana, S.; Romera-Paredes, B.; Vineet, V.; Su, Z.; Du, D.; Huang, C.; and Torr, P. H. 2015. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1529–1537.

Ziegler, Z. M., and Rush, A. M. 2019. Latent normalizing flows for discrete sequences. *arXiv preprint arXiv:1901.10548*.