Structured Conditional Continuous Normalizing Flows for Efficient Amortized Inference in Graphical Models

Christian Weilbach  Boyan Beronov  William Harvey  Frank Wood
Department of Computer Science, University of British Columbia
{weilbach, beronov, wagh, fwood}@cs.ubc.ca
University of British Columbia, 2329 West Mall Vancouver, BC Canada V6T 1Z4

Abstract

We exploit minimally faithful inversion of graphical model structures to specify sparse continuous normalizing flows (CNFs) for amortized inference. We find that the sparsity of this factorization can be exploited to reduce the numbers of parameters in the neural network, adaptive integration steps of the flow, and consequently FLOPs at both training and inference time without decreasing performance in comparison to unconstrained flows. By expressing the structure inversion as a compilation pass in a probabilistic programming language, we are able to apply it in a novel way to models as complex as convolutional neural networks. Furthermore, we extend the training objective for CNFs in the context of inference amortization to the symmetric Kullback-Leibler divergence, and demonstrate its theoretical and practical advantages.

1 Introduction

Continuous normalizing flows (CNFs) are a flexible class of density estimators (Grathwohl et al., 2018) consisting of learnable ordinary differential equation systems (Chen et al., 2018). While state-of-the-art in terms of density estimation, CNFs require more computation than other density estimators and have proven hard to scale up to high dimensions, limiting them to comparatively low dimensional problems. Since inference amortization (Gershman and Goodman, 2014), is a necessary ingredient to scale probabilistic machine learning methods up to many real-world applications, we focus our work on this problem domain.

To better integrate CNFs with probabilistic modeling, we make use of a programming language design inspired mindset. We have a particular focus on performing Bayesian inference in probabilistic programs (van de Meent et al., 2018), i.e. stochastic computer simulations which are formally interpreted as statistical models, although our main contributions are applicable to any statistical process which can be expressed as a graphical model. By automatically translating between representations of these inference models, we can use explicit structural information symmetries in the models to guide the flow of information during inference. Our contributions are as follows:

1. We describe a novel class of neural networks that incorporate structural constraints of statistical models through the sparsity of their weight matrices. By deriving this structure from a formal specification of an inverse problem, we train continuous normalizing flows as amortized inference artifacts and show that, depending on the model, they need less than half the number of floating point operations (FLOPs). This construction performs as well as, or better than, a state-of-the-art neural network from FFJORD (Grathwohl et al., 2018). We additionally provide a flexible way to augment the dimension of the flows, while assigning meaning to each dimension in terms of a directe mapping to the graphical model.

2. The continuous flows yield probability distributions which are efficient both to sample from and to compute densities with respect to. We show that in the amortized inference setup, the loss can be extended to a symmetrized Kullback-Leibler divergence, which improves the training of amorti-
Structured Conditional Continuous Normalizing Flows

3. Finally, we formalize our translation process as a probabilistic programming language compiler backend and apply it to the inverse problem of dimension-increasing image deconvolution, by automatically inverting a standard convolution operator from the deep learning literature.

2 Background

2.1 Probabilistic Programming

Probabilistic programming allows users to express a joint density, \( p(x, z) = p(x|z)p(z) \), as a generative procedure denoted in a programming language such as Python (Bingham et al., 2018) or Clojure (Tolpin et al., 2016). The code denotes a generative model, which transforms samples from a prior \( p(z) \) into a distribution over observed data via the likelihood \( p(x|z) \). Given such a generative procedure, we tackle the problem of inferring the posterior \( p(z|x) \). We point the interested reader to van de Meent et al. (2018) to learn more about the underlying probabilistic programming language.\(^1\) However, our work does not require a deep understanding of this background: our improvements can be understood as automatic translation from structural knowledge about a generative model into constrains on the inference procedure. In particular, our programming language can be thought of as an expressive syntax to denote graphical models (Koller and Friedman, 2009) over continuous densities. For example, the left column of Figure 1 displays a probabilistic program, written in pseudocode, which is compiled (translated) to the graphical model in the second column.

2.2 Faithful Model Inversion

Provided the graphical structure which captures the independences between latent variables, we apply the faithful inversion algorithm of Webb et al. (2018). This returns the structure of an ‘inverted’ stochastic model, mapping from observations to distributions over the latent variables. In particular, the greedy inversion algorithm returns a structure with approximately minimal number of new edges required to faithfully capture the dependency structure of the inverse model. As an example, the third column of Figure 1 shows a faithful inverse of the graphical model in the second column.

\(^1\)The implementation can be found at https://github.com/plai-group/daphne

2.3 Amortized Inference

Amortized inference techniques (Gershman and Goodman, 2014; Ritchie et al., 2016) yield efficient posterior approximations, \( q(z|x) \approx p(z|x) \), typically by maximizing a variational evidence lower bound (ELBO or \( \mathcal{L} \)) on \( p(x) \) (Blei et al., 2017; Kingma and Welling, 2013). This corresponds to minimizing the reverse Kullback-Leibler divergence (KL) (Bishop, 2006):

\[
\mathcal{L}[p; q] = - \int q(z|x) \log \frac{p(x, z)}{q(z|x)} dz \approx \mathcal{L}[p; q_\Phi] = \mathcal{L}[p; q] - \mathcal{L}[q; q_\Phi].
\]

In order to learn proposals that are good for many \( x \), an expectation of this bound is taken over some distribution \( x \). Training neural artifacts, \( \Phi \), to parameterize \( q(z|x) = q_\Phi(z|x) \) trades off upfront computation against cheap approximation of \( p(z|x) \) at inference time.

One particular type of amortization, which has been coined inference compilation for probabilistic programs (Le et al., 2017; Paige and Wood, 2016), involves learning \( \Phi \) to minimize a variational loss with the following gradient:

\[
\nabla_\Phi \mathcal{L}[p; q_\Phi] = - \mathbb{E}_{p(x,z)} \left[ \nabla_\Phi \log q_\Phi(z \mid x) \right],
\]

where \( x \) are the observations, and \( z \) the latent variables that we learn a distribution over. This is the same loss used in the sleep-phase of the wake-sleep algorithm (Le et al., 2019). The expectation is over synthetically sampled data generated from the joint distribution \( p(x, z) \) of the program. Minimizing this loss corresponds to minimizing the forward KL, \( k_{DKL}(p(\cdot|x) || q_\Phi(\cdot|x)) \). The learned artifact can be used to speed up asymptotically exact inference in the generative model by using \( q_\Phi(z|x) \) as a proposal distribution for Sequential Importance Sampling (Doucet and Johansen, 2009).

2.3.1 Continuous Normalizing Flows

A neural ordinary differential equation (ODE) system (Chen et al., 2018) can be defined in this setting by a reference prior \( q^0(z_0) \) and a deterministic flow-defining neural network \( f_\theta \) on latent particles \( z \):

\[
\frac{dz_t}{dt} = f_\theta(z_t, t; x).
\]

Conditioning is achieved by passing \( x \) into the network as an additional constant input. The numerical computation at inference time constitutes the integration of independent particle trajectories with dynamics given by Equation (3), from initial conditions \( z_0 \sim q^0 \) at time \( t = 0 \) towards the posterior approximation at \( t = 1 \), using a standard ODE solver. In order to obtain a normalized distribution at the end of the flow, the
Figure 1: A generative model denoted in (1) is compiled to the graphical model in (2). This is structurally inverted to yield (3), which is translated into a sparse neural network parametrizing a continuous normalizing flow (4). This learned flow then provides an approximation of the posterior, \( p(z|x) \), of the model in (1). The 4 weight matrices of the neural network architecture are shown as Hinton diagrams with positive weights as white squares and negative weights as black squares, and size proportional to the magnitude. Note the sparsity of the weights, due to the sparse structure of the inverse model in (3). For clarity, the augmenting dimensions, as described in Section 3.1.2 and Figure 2, are not shown.

3 Methods

In this section we describe our adapted amortization methods for continuous normalizing flows. First we will explain how the flow can be structured and adapted to be more efficient in Section 3.1, and then we describe how we train the network to minimize a symmetrized Kullback-Leibler (KL) divergence in Section 3.2.

3.1 Structured Flows

3.1.1 Sparse Neural ODE

Layers of the neural network of \cite{Grathwohl2018} take the form of

\[
\hat{z}_t = \sigma(W\hat{z}_t \odot \eta_1(t)) + b \odot \eta_2(t). 
\]

where \( (\nabla_z \cdot f) \) denotes the divergence of \( f \) or, equivalently, the trace of the Jacobian of \( f \) \cite{Chen2019}.

There are three main algorithmic advantages of this approach to density estimation: its intrinsic parallelism between independent particles; the fact that the flow transformation is invertible at equal cost simply by executing the integration in the opposite direction; and freedom in neural architecture choice. The second advantage makes it possible to cheaply calculate the log probability, and contrasts with alternative approaches such as invertible residual networks \cite{Behrmann2019} and non-continuous normalizing flow architectures \cite{Rezende2015}. The third advantage, compared to non-continuous flow architectures, is that CNFs do not impose any constraints on the internal structure of the neural network to retain a computable Jacobian, a freedom we exploit to restrict the sparsity structure as described in the following section.

3.1.2 Augmented Normalizing Flows

Recently, it has been demonstrated that adding auxiliary dimensions to the state space of flows can ease the
In particular, using the symmetrized KL divergence, our objective for a variational posterior $q_\Phi$ is

$$\mathcal{L}[p](q_\Phi) = \mathbb{E}_{x \sim p_X} \left\{ \mathcal{D}_{\text{sym}} \{ p(\cdot | x) \| q_\Phi(\cdot | x) \} \right\}$$

$$= \frac{1}{2} \mathbb{E}_{x \sim p_X} \left\{ \frac{1}{2} \mathbb{E}_{z \sim q_\Phi(\cdot | x)} \left( \frac{\ln \frac{p(z, x)}{q_\Phi(z | x)}}{p(z, x)} \right) \right\} + \mathbb{E}_{x \sim p_X} \left\{ \mathbb{E}_{z \sim q_\Phi(\cdot | x)} \left[ \frac{\ln \frac{q_\Phi(z | x)}{p(z, x)}}{p(z, x)} \right] \right\}.$$

Equation (7) uses the fact that the evidence $p_X$ is a constant w.r.t. $z$, cancelling out among the two terms and rendering the full objective tractable. Furthermore, if $p$ or $q$ have little mass in some region of the probability space, the respective KL divergence will be weakly affected by the other distribution, leading to a weak learning signal from this region (Bishop 2006). While each KL vanishes only when $p$ and $q$ match perfectly, Section 4.1 demonstrates a significant benefit from combining both terms in $\mathcal{D}_{\text{sym}}$ as complementary learning signals. We hence arrive at the loss gradient

$$\nabla_\Phi \mathcal{L}[p](q_\Phi) = \frac{1}{2} \left\{ \mathbb{E}_{(z, x) \sim p} \left( -\nabla_\Phi \ln q_\Phi(z | x) \right) + \mathbb{E}_{x \sim p_X} \left( \nabla_\Phi \mathbb{E}_{z \sim q_\Phi(\cdot | x)} \left[ \ln \frac{q_\Phi(z | x)}{p(z, x)} \right] \right) \right\}.$$

A Monte Carlo approximation of the first term of Equation (8) can be obtained by taking samples $z, x$ from the joint model $p$ and propagating $z$ along the flow, conditioned on $x$, towards the reference distribution $q^0$ (deconditioning flow) — thus we can maximize the log-probability of $q$ for these samples. Additionally, by initializing particles at the reference distribution $q^0$ (conditioning flow) and making use of the fact that the dynamics defined by the same observations $z$ can be reused in the opposite direction of the ODE system, variational posterior samples $\hat{z}$ in the second term of Equation (8) can be generated. Since the inner expectation of $\hat{z}$ is reparametrized by $q^0$ in form of standard normal distributions we can differentiate through both directions of the flow in one step. To summarize, embracing a differentiable composition of functions with the flow integrator provides a flexible, well-defined optimization setup as described in Algorithm 1 that is an extension of the algorithm in Grathwohl et al. (2018).

### 3.3 Joint Space Normalization

Before training, we apply a change of variables on $p(x, z)$ to normalize the moments of its marginals to be the same as those of $q^0$, i.e., zero mean and unit variance. These moments are estimated by a sample of 10,000 draws from the joint distribution. This transformation avoids the flow facing inputs that could be...
scaled arbitrarily and could render its training unstable, and is denoted with $S$ in Algorithm 1.

4 Experiments

In this section we show an ablation study on the arithmetic circuit model in Figure 1. First we describe the setup and show that the resulting flow matches the marginals of the joint distribution in Section 4.1. We then demonstrate that the faithful inversion structure makes training and runtime faster and more efficient in Section 4.2. We evaluate our loss functional in Section 4.3. Finally, we demonstrate that our setup allows us to invert the convolution operation as it is used in convolutional neural networks. We describe the structure of the inverse and perform a qualitative study that in Section 4.4.

4.1 Synthetic Model: Arithmetic Circuit

The arithmetic circuit we use is defined in the first column of Figure 1. The link functions contain combinations of multiplication, addition and tanh operators, similar to primitives used in deterministic neural networks. The priors on $z_0$ and $z_1$ are heavy-tailed, making them much more challenging to invert than a Gaussian distribution would be. The standard deviations around the tanh expressions are 0.1, demanding a high degree of precision in the stochastic, non-linear inversion. We first test on this model, because it contains operations which could serve as the building blocks for large-scale models, and because it has a non-trivial dependency structure and is challenging enough for the forward KL objective of FFJORD (Grathwohl et al. 2018) to perform badly.

We compare the marginals of the faithfully inverted sparse flow for $q(z|x)p(x)$ with $p(x,z)$ in Figure 3, providing a loss-independent consistency check. Each node in the graphical model has been augmented by 10 dimensions, as described in Section 3.1.2, and we have optimized for 10,000 iterations with a batch size of 100 samples. The marginal distributions match well, corresponding to a final expected symmetric KL loss of about 0.1 nats.

4.2 Sparsity Pattern

Next, we are interested in how our sparsity structure (Section 3.1) affects the learned flow. In Figure 4, we
Figure 4: Effect of architectural choices on the expected symmetrized KL training loss for the experiment in Section 4.1. We plot the median (smoothed in time) and a confidence band between the 16th and 84th percentiles for 10 runs. The learning rate is reduced from $10^{-2}$ to $10^{-3}$ at iteration 2500 and to $5 \times 10^{-4}$ at iteration 4000. Our method learns faster in the beginning, but converges to similar results as fully connected models in the end. The random sparse baseline quickly saturates and experiences high variance.

Figure 5: Effect of sparsity patterns on the numerical stability and computation time during training, more iterations being worse. We again plot the median and a confidence band between the 16th and 84th percentiles over 10 runs. The faithfully inverted neural net leads to a numerically better conditioned flow than both the fully connected variant and the FFJORD baseline in all three cases.

4.2.1 Stability and Tolerance of the Adaptive Integrator

We have also analyzed the behavior of the Runge-Kutta integrator used in CNFs during training in Figure 4. The measurements of conditioning taken across training are equivalent to the cost at inference time. Our sparsely structured flow requires the fewest iterations. The number of FLOPS at runtime is the product of the number of steps taken by the solver and the FLOPS of each forward pass in the neural network. Each weight in the network is multiplied once with each input dimension in a forward pass. Ignoring the small number of bias parameters, we conclude that we need less than half as many FLOPs as the fully connected or FFJORD variants. Additional plots involving the deconditioning pass can be found in Appendix A.

4.3 Objective Function

Figure 6 shows a comparison of the different losses described in Section 3.2. The reverse KL-based loss, corresponding to the second term in Equation (8), was found to be capable of training simpler models, such as small Gaussian state space models. However, it had consistently higher variance than the forward KL and was not at all sufficient for training on the arithmetic circuit we consider, as Figure 6 shows. The forward KL, the standard loss introduced with CNFs (Grathwohl et al., 2018), provides a learning signal on the task, but saturates quickly with a symmetric KL of about 100 nats. The symmetrized KL, on the other hand, learns faster from the start and keeps improving to below 10 nats. This is a crucial improvement, since the forward KL only optimizes $q$ to be a density estimator for $p(z|x)$, while the reverse KL optimizes the sampling behavior of $q$ as well. Our experiment shows that such a CNF...
Figure 6: Effect of optimizing different terms of the objective function for the experiment in Section 4.1. Optimizing the reverse KL term alone does not provide a sufficient learning signal while the forward KL term does provide a good training signal. By combining both update directions in one step, we improve our symmetric KL objective on the arithmetic circuit by more than an order of magnitude. We again plot the median and a confidence band between the 16th and 84th percentiles over 10 runs. See also Appendix B.

4.4 Deconvolution

Convolutional neural network architectures are successfully applied in a wide range of applications and have revolutionized the field of computer vision (LeCun et al., 2015). Their deterministic mapping from images to low-dimensional outputs is a well-defined operation, but the inverse is not uniquely defined, i.e. one label can map to many images. Nonetheless, it is common to apply deterministic (de)convolution to design generative architectures for images (Donahue et al., 2016), but it is generally understood that there are problems like checkerboard artifacts (Odena et al., 2016). Deterministic upsampling is alternatively used to increase dimensionality (Karras et al., 2019) while circumventing checkerboard effects, but suffers from the same underdetermination.

We explore a more principled approach to deconvolution, taking the convolutional operator of convolutional neural network classifiers (Dumoulin and Visin, 2016) and using our faithful inversion to derive a stochastic inverse, which is a natural computational interpretation of the non-deterministic inverse. Describing the convolutional tensor operator in a probabilistic programming language we can generate the inverse structure automatically as shown in Figure 8. This would be extremely tedious to do manually, and therefore has not been studied previously.

In Figure 9 we explore the amortized inversion of convolutional filters and generate samples \( z_{in} \in \mathbb{R}^{9 \times 9} \) of randomly cropped patches from the MNIST dataset (LeCun et al., 1998), filters \( x_{filter} \in \mathbb{R}^{3 \times 3} \) and convolved output \( x_{out} \in \mathbb{R}^{4 \times 4} \) as computed by the program in Figure 7. We sample each of the filter pixels \( x_{filter} \sim \mathcal{N}(0; 1) \). Filters of this size are commonly used in convolutional architectures like LeNet (LeCun et al., 1998).

Since we do not have access to prior over patches, and do not want to learn another density estimator for the prior at a similar complexity as estimating our posterior, we only use the forward KL (eq. 2). The resulting artifact amortizes over all possible filters, and hence only needs to be trained once to invert convolutions of this form. The reconstructed outputs in Figure 9 match the output we have conditioned on closely in all cases.

5 Related Work

5.1 Flows

Recently, a unified perspective on the powerful class of residual networks (ResNets) (He et al., 2016) in deep learning and ODEs has been established (Ruthotto and Haber, 2019; Ciccone et al., 2018). In Chen et al.
Structured Conditional Continuous Normalizing Flows

(2019) the dynamics of the probability density \( q(\cdot, t) \) are connected to the continuity equation and their work provides a continuous normalizing flow for particle filter methods. There are also new approaches to approximate stochastic differential equations (SDEs) (Hegde et al., 2019).

**Conditional Normalizing Flows**

Alternative approaches to conditioning using the mechanism of an embedding VAE have been explored in (Grathwohl et al., 2018). In (Trippe and Turner, 2018) the parameters of the neural network itself are conditioned on the observation. The approach of conditioning CNFs in this way could also be translated back to invertible residual networks (Behrmann et al., 2019), but inversion in this setting is more expensive and a symmetrized KL would not work without fixpoint searches. Inverse autoregressive flows (Kingma et al., 2016) are another popular way of conditioning normalizing flows, but autoregressive models have the drawback of requiring a sequential roll-out over each dimension of the latent space.

**5.2 Graph Networks**

The application of deep learning to graphical modeling is a well-established field (Bronstein et al., 2017, Wu et al., 2019). Our work can be understood as graph normalizing networks (Liu et al., 2019) that implement a neural network based message passing algorithm along an underlying graph structure. This approach has recently been extended to continuous normalizing flows (Deng et al., 2019), including inference in probabilistic graphical models. We build on the same intuition and continuously propagate information from neighboring nodes, but also project the graphical model structure onto the neural network and apply it to an extended amortized inference setting.

**6 Conclusion**

In this paper we demonstrated that by systematic integration of structural knowledge, we can improve amortized inference for complex continuous graphical models denoted as probabilistic programs. In particular, we have structured the neural networks used in continuous normalizing flows in a way that increases efficiency and makes it possible to adaptively augment the flow with auxiliary dimensions. Additionally, we have extended the optimization loss used to train continuous normalizing flows and shown that it significantly improves training. We think that integration of discrete variables and model learning are interesting future extensions of our work. Our approach can be generalized to integrate more knowledge about optimization of dynamical systems, information geometry and domain specific languages that express more prior knowledge about problem structure. In particular, we plan to use information theoretic measures during training to implement an online adaptive scheme to augment the flow with new dimensions. This could simplify the expensive and challenging process of neural architecture search (Elsken et al., 2019) while scaling up our networks.
Acknowledgments

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, Compute Canada, Intel, and DARPA under its D3M and LWLL programs.

References


