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, wsgh, 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), for example in the form of variational auto-encoders (Kingma and Welling, 2013), 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-

Proceedings of the 23rdInternational Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

zation artifacts significantly.

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 $\mathcal{E}$) on $p(x)$ (Blei et al., 2017; Kingma and Welling, 2013). This corresponds to minimizing the reverse Kullback-Leibler divergence (KL) (Bishop, 2006):

$$\begin{aligned}\mathcal{E}[p](q; x) &= \mathop{\mathbb{E}}_{z \sim q(\cdot|x)}[\log p(x, z) - \log q(z|x)] \\ &= \log p(x) - \mathcal{D}_{\mathrm{KL}}\{q(\cdot|x) \,||\, p(\cdot|x)\}. \end{aligned} \quad (1)$$

In order to learn proposals that are good for many x, an expectation of this bound is taken over some distribution of x. Training neural artifacts, Φ, 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 Φ to minimize a variational loss with the following gradient:

$$\nabla_\Phi \mathcal{L}[p](q_\Phi) = - \mathop{\mathbb{E}}_{p(x,z)}[\nabla_\Phi \log q_\Phi(z \mid x)], \quad (2)$$

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, $\mathcal{D}_{\mathrm{KL}}\{p(\cdot|x) \,||\, q_\Phi(\cdot|x)\}$. 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_Φ on latent particles z:

$$\frac{\mathrm{d}}{\mathrm{d}t} z_t = f_\Phi(z_t, t; x). \quad (3)$$

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

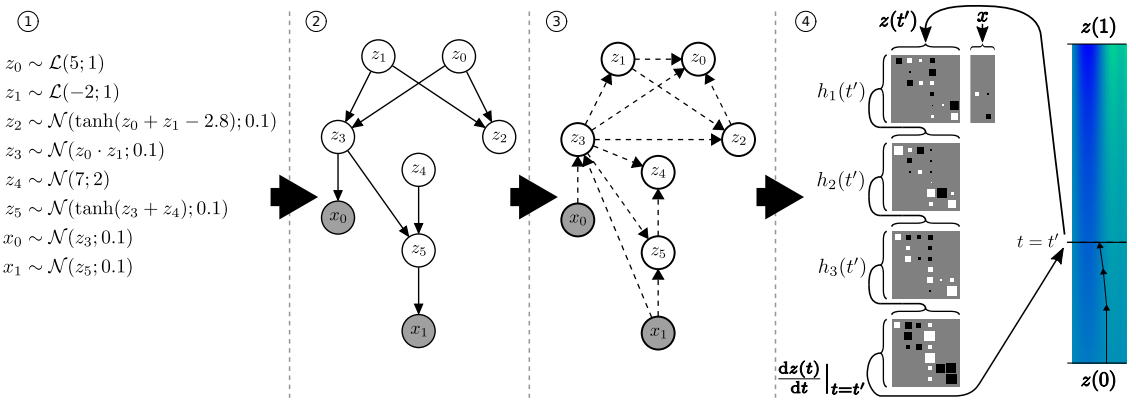


Figure 1: A generative model denoted in (1) is compiled to the graphical model in (2). This is structurally inverted to yield (3), which is translated into a sparse neural network parametrizing a continuous normalizing flow (4). This learned flow then provides an approximation of the posterior, $p(z|x)$, of the model in (1). The 4 weight matrices of the neural network architecture are shown as Hinton diagrams (Hinton and Shallice, 1991) 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.

log-probability of each particle must be integrated as follows alongside the particle dynamics:

$$\frac{\mathrm{d}}{\mathrm{d}t} \ln q_\Phi(z_t, t) = -\nabla_z \cdot f_\Phi(z_t, t; x). \tag{4}$$

where $(\nabla_z \cdot f)$ denotes the divergence of f or, equivalently, the trace of the Jacobian of f (Chen et al., 2019).

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 (Behrmann et al., 2019) and non-continuous normalizing flow architectures (Rezende and Mohamed, 2015). 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 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 (Grathwohl et al., 2018) take the form of

$$h(\hat{z}, t) = \sigma\big\{(W\hat{z} \odot \eta_1(t)\big\} + b \odot \eta_2(t)\ . \tag{5}$$

σ is the activation function tanh, W are the weights, b is a bias, and $\eta_{1,2}$ are time dependent linear gating functions. In order to constrain the connectivity of each layer of our neural network

$$f_\Phi(z, t; x) = \left(h_{\Phi_L}(\cdot, t) \circ \cdots \circ h_{\Phi_1}(\cdot, t)\right)(z \oplus x)$$

to respect the necessary statistical independence structure, our contribution is to mask its weight matrix with the adjacency H of the minimally faithful inverted graphical model, i.e., the output of each layer l reads

$$h_{\Phi_l}(\hat{z}, t) = \sigma\big\{(W_l \odot H)\hat{z} \odot \eta_{l,1}(t)\big\} + b_l \odot \eta_{l,2}(t). \tag{6}$$

A simplified version of this architecture is shown in Panel (4) of Figure 1. Here $\hat{z}_0 = z \oplus x$ is a concatenation before the first layer, and $\hat{z}_l = h_{l-1}(\hat{z}_{l-1}, t)$ in subsequent layers, such that each column $\hat{z}_0^i, \ldots, \hat{z}_L^i$ of neurons across layers corresponds to a node i in the graphical model.

3.1.2 Augmented Normalizing Flows

Recently, it has been demonstrated that adding auxiliary dimensions to the state space of flows can ease the

learning task (Dupont et al., 2019) by reducing the topological constraints on the dynamics. This is comparable to the fact that feature spaces of higher dimensions are more likely to render a dataset linearly separable (Bishop, 2006). In our framework, the augmentation needs to additionally respect the probabilistic nature of the dynamics, including the effect on Equation (4). To do so, we add to each node in the graphical model a flexible number of nuisance variables, maintaining the same form for the reference distribution as q^0. At training time, these dimensions are constrained to have the same posterior as the prior, while at inference time, they are marginalized out, i.e., simply ignored. By choosing the prior and posterior of augmenting dimensions to be identical in distribution, nuisance variables are encouraged to provide a purely internal communication channel to aid in the inference of latent variables. Figure 2 illustrates a possible augmentation for the subgraph $\{x_1, z_4, z_5\}$ of panel 3 in Figure 1, such that z_4 is augmented by y_4^1, while z_5 is augmented by $y_5^{1,2}$. Note that by building cliques at the fine level among each latent variable and its associated nuisance variables, the original directed faithful inverse structure is preserved at the coarse level, while obtaining maximal connectivity in the network and still relating each neuron to a unique node in the graphical model. All fine-level nodes in a clique point to all subnodes of any adjacent clique.

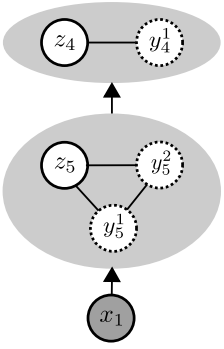


Figure 2

3.2 Symmetrized KL Divergence

As noted in Section 2.3, the objective in variational inference is usually the evidence lower bound corresponding only to the reverse Kullback-Leibler (KL) divergence, whereas inference compilation uses the expectation of the forward KL under the marginal data distribution of the generative model, $p_X(x) = \int_Z p(z, x)\,\mathrm{d}z$. In our setting, we can make use of the generative model p, as well as the learned flow q_Φ in both directions. We therefore combine the two approaches and propose the use of the expected symmetrized KL divergence, $\mathcal{D}_{\mathrm{sym}}\{p, q\} = \frac{1}{2}\left(\mathcal{D}_{\mathrm{KL}}\{p, q\} + \mathcal{D}_{\mathrm{KL}}\{q, p\}\right)$, as an objective for inference amortization on continuous normalizing flows. $\mathcal{D}_{\mathrm{sym}}$, also called the Jeffrey's divergence (Nielsen, 2010), is theoretically appealing because we can approximate its expectation, while both the forward and the reverse KL divergence can only be evaluated up to a constant shift by the evidence. We can therefore meaningfully compare convergence on different models.

In particular, using the symmetrized KL divergence , our objective for a variational posterior q_Φ is

$$\begin{aligned}\mathcal{L}[p](q_\Phi) &= \mathop{\mathbb{E}}_{x\sim p_X}\left\{\mathcal{D}_{\mathrm{sym}}\{p(\cdot \mid x)\,||\,q_\Phi(\cdot \mid x)\}\right\} \\ &= \frac{1}{2}\mathop{\mathbb{E}}_{x\sim p_X}\left\{\mathop{\mathbb{E}}_{z\sim p(\cdot|x)}\left(\ln\frac{p(z,x)}{q_\Phi(z \mid x)}\right) + \right. \\ &\qquad \left. \mathop{\mathbb{E}}_{z\sim q_\Phi(\cdot|x)}\left(\ln\frac{q_\Phi(z \mid x)}{p(z,x)}\right)\right\}. \quad (7)\end{aligned}$$

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 $\mathcal{D}_{\mathrm{sym}}$ as complementary learning signals. We hence arrive at the loss gradient

$$\begin{aligned}&\nabla_\Phi\,\mathcal{L}[p](q_\Phi) \\ &= \frac{1}{2}\left\{\mathop{\mathbb{E}}_{(z,x)\sim p}(-\nabla_\Phi \ln q_\Phi(z \mid x)) + \right. \\ &\qquad \left. \mathop{\mathbb{E}}_{x\sim p_X}\left(\nabla_\Phi \mathop{\mathbb{E}}_{\hat{z}\sim q_\Phi(\cdot|x)}\left[\ln\frac{q_\Phi(\hat{z} \mid x)}{p(\hat{z},x)}\right]\right)\right\}. \quad (8)\end{aligned}$$

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 x 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

Algorithm 1 Training a structured conditional flow for inference amortization

Input: joint model p, reference distribution q^0, normalizing transform S on latents z.
Output: conditional flow f_Φ.

while not *converged*(Φ) **do**
  $(z, x) \sim p$ ▷ sample from joint distribution
  $\langle \bar{z}_0, \Delta_{\ln q_\Phi(S(z)|x)} \rangle \leftarrow \mathit{odeint}_1^0[f_\Phi(\cdot; x)]\big(\langle S(z), 0\rangle\big)$ ▷ *deconditioning*: jointly integrate eq. $\langle(3), (4)\rangle$
  $\ln q_\Phi(z \mid x) \leftarrow \ln q^0(\bar{z}_0) - \Delta_{\ln q_\Phi(S(z)|x)} - \ln \left|\frac{\mathrm{d}S}{\mathrm{d}z}(z)\right|$ ▷ change of variables: $\mathrm{CNF}^{-1} \circ S$

  $z_0 \sim q^0$ ▷ sample from reference distribution
  $\langle \check{z}, \Delta_{\ln q_\Phi(\check{z}|x)} \rangle \leftarrow \mathit{odeint}_0^1[f_\Phi(\cdot; x)]\big(\langle z_0, 0\rangle\big)$ ▷ *conditioning*: jointly integrate eq. $\langle(3), (4)\rangle$
  $\hat{z} \leftarrow S^{-1}(\check{z})$
  $\ln q_\Phi(\hat{z} \mid x) \leftarrow \ln q^0(z_0) + \Delta_{\ln q_\Phi(\check{z}|x)} + \ln \left|\frac{\mathrm{d}S^{-1}}{\mathrm{d}\check{z}}(\check{z})\right|$ ▷ change of variables: $S^{-1} \circ \mathrm{CNF}$

  $\Phi \leftarrow \mathit{update}\big(\Phi, \nabla_\Phi \mathcal{L}[p](q_\Phi)\big)$ ▷ MC estimate of (8) using loss terms from lines 6,11
end while

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.

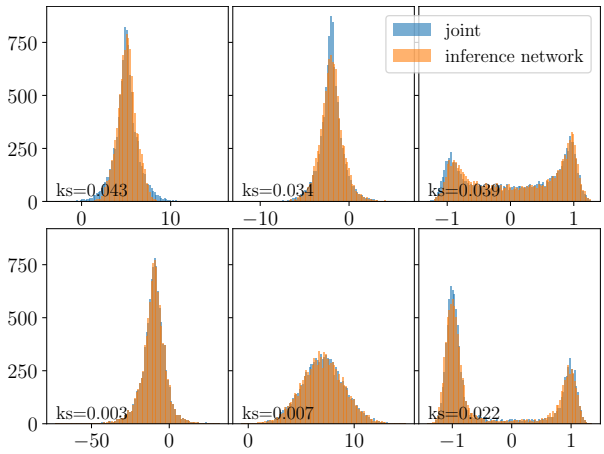


Figure 3: Histogram comparison between samples from the joint and the learned conditioning flow, for the arithmetic circuit from section 4.1 with 10,000 samples. The marginals are listed in rows from z_0 in the top left to z_5 in the bottom right. The Kolmogorov-Smirnov test statistic is printed in the bottom left of each histogram. We can see that the inference network has problems matching the heavy tails of z_0. All other latents are matched almost perfectly.

4.2 Sparsity Pattern

Next, we are interested in how our sparsity structure (Section 3.1) affects the learned flow. In Figure 4 we

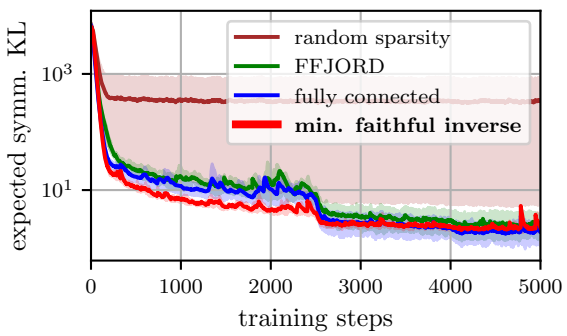
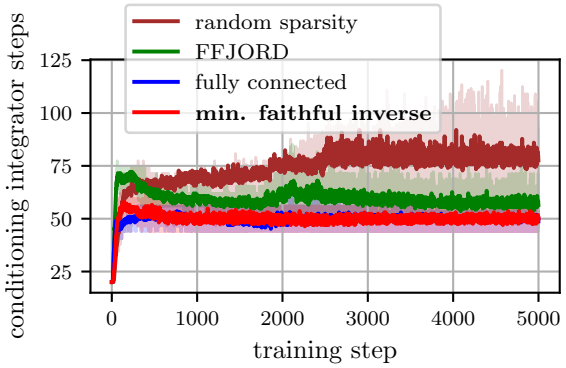


Figure 4: Effect of architectural choices on the expected symmetrized KL training loss for the experiment in Section 4.1. We plot the median (smoothed in time) and a confidence band between the 16th and 84th percentiles for 10 runs. The learning rate is reduced from 10^{-2} to 10^{-3} at iteration 2500 and to $5 * 10^{-4}$ 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.



(a) Conditioning

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.

compare our faithful inverse structured flow to three baselines: our setup with a fully connected flow; random sparse structure with the same edge density; and the architecture used in FFJORD (Grathwohl et al., 2018). FFJORD projects the low dimensional flow onto fully connected hidden layers of size 64 internally. All other hyperparameters are the same. The sparse and faithfully inverted networks both have 10 dimensions as augmentation, yielding a flow of 66 dimensions. The flow with full connectivity of 18679 weight parameters is contrasted with minimally faithfully inverted sparse connectivity, counting only 7725 parameters, whereas the FFJORD neural network baseline has 17801 parameters.

The network with faithfully inverted structure, despite having less than half the number of parameters, learns faster, while the fully connected network only catches up at the end. The FFJORD baseline, with a structure similar to the fully connected network, behaves slightly worse. The flow with a random sparse connectivity has significantly worse performance, due to the lack of a structure reflecting the statistical dependencies. We conclude that the faithfully inverted structure is a strict improvement both in terms of training convergence and, using a sparse matrix implementation, the number of floating point operations at runtime.

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 5. 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

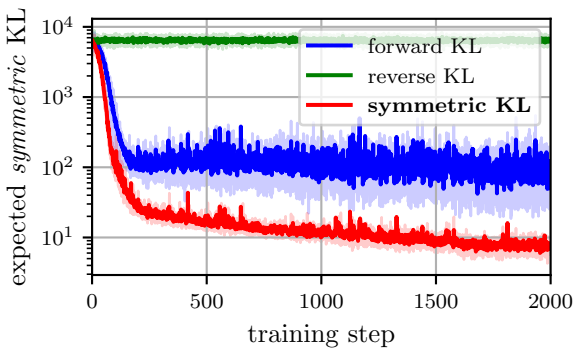


Figure 6: Effect of optimizing different terms of the objective function for the experiment in Section 4.1. Optimizing the reverse KL term alone does not provide a sufficient learning signal while the forward KL term does provide a good training signal. By combining both update directions in one step, 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.

can only be trained with the symmetric KL. For this run, we have used an augmentation of 5 dimensions for each latent variable, while the other parameters were the same as in the previous result. The benefits of the symmetric KL loss were consistent over all of our experiments, and can also be seen on the Gaussian State Space model in Appendix D.

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

```
inputs=sample_MNIST_8x8_patch()
for i in range(3):
    for j in range(3):
        sample(Normal(0, 1), obs=filter[i, j])

def conv2d(inputs, filters, stride):
    return [[dot(x[sy, sx], filters) + bias
             for sx in slices_x(inputs, stride)]
            for sy in slices_y(inputs, stride)]

output = conv2d(inputs, filters, 0, 2)
for i in range(4):
      for j in range(4):
          sample(Normal(0, 1), obs=output[i, j])
```

Figure 7: Pseudocode for the convolutional operator we invert. We implement all tensor operations, including *dot*, *slices_x* and *slices_y* in our language to make them transparent to our inversion algorithm.

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 $z_{\text{in}} \in \mathbb{R}^{9\times9}$ of randomly cropped patches from the MNIST dataset (LeCun et al., 1998), filters $x_{\text{filter}} \in \mathbb{R}^{3\times3}$ and convolved output $x_{\text{out}} \in \mathbb{R}^{4\times4}$ as computed by the program in Figure 7. We sample each of the filter pixels $x_{\text{filter}} \sim \mathcal{N}(0; 1)$. 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.

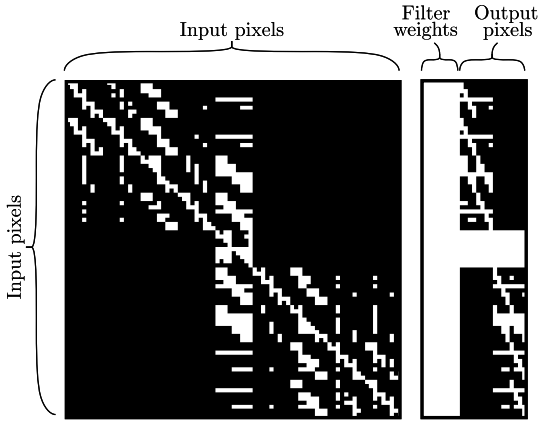


Figure 8: The faithful inverse structure for a 2D convolution. Black denotes the parts that are zeroed out in the adjacency matrix of pixels, white the connected parts. All input pixels x depend on all filter pixels, but only on the outputs that they are influencing. Pixels in the middle of the image depend on all output pixels. The faithful inversion algorithm also automatically infers dependencies between pixels.

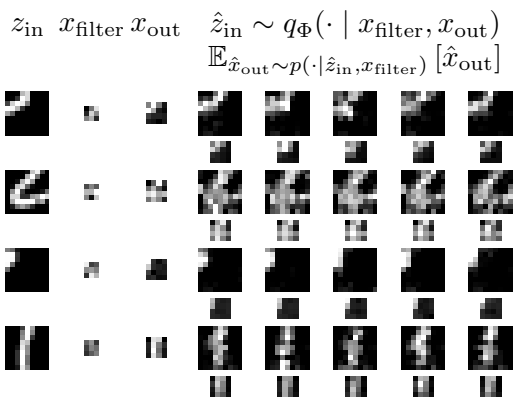


Figure 9: Visualization of our stochastic deconvolution. We take patches of MNIST images (left column), and sample filters from the prior (second column). Convolution of each image patch and filter pair, along with the addition of Gaussian noise, produces the outputs in the third column. The remaining columns show 5 samples from q, conditioned on each filter and output (odd rows; compare with z_{in}), and reconstructions of the output given these (even rows; compare with x_{out}).

(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

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J. (2019). Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 573–582.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2018). Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 6572–6583.

Chen, X., Dai, H., and Song, L. (2019). Particle flow Bayes' rule. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1022–1031.

Ciccone, M., Gallieri, M., Masci, J., Osendorfer, C., and Gomez, F. J. (2018). NAIS-net: Stable deep networks from non-autonomous differential equations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 3029–3039.

Deng, Z., Nawhal, M., Meng, L., and Mori, G. (2019). Continuous graph flow for flexible density estimation. *CoRR*, abs/1908.02436.

Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *CoRR*, abs/1605.09782.

Doucet, A. and Johansen, A. M. (2009). A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *CoRR*, abs/1603.07285.

Dupont, E., Doucet, A., and Teh, Y. W. (2019). Augmented neural ODEs. In *Advances in Neural Information Processing Systems*, pages 3134–3144.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Gershman, S. and Goodman, N. (2014). Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36.

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. K. (2018). FFJORD: free-form continuous dynamics for scalable reversible generative models. *CoRR*, abs/1810.01367.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778.

Hegde, P., Heinonen, M., Lähdesmäki, H., and Kaski, S. (2019). Deep learning with differential Gaussian process flows. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, pages 1812–1821.

Hinton, G. E. and Shallice, T. (1991). Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological Review*, 98(1):74–95.

Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410.

Kingma, D. P., Salimans, T., Józefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improving variational autoencoders with inverse autoregressive flow. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4736–4744.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *CoRR*, abs/1312.6114.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models - Principles and Techniques*. MIT Press.

Le, T. A., Baydin, A. G., and Wood, F. (2017). Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pages 1338–1348, Fort Lauderdale, FL, USA. PMLR.

Le, T. A., Kosiorek, A. R., Siddharth, N., Teh, Y. W., and Wood, F. (2019). Revisiting reweighted wake-sleep for models with stochastic control flow. In Globerson, A. and Silva, R., editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, page 373. AUAI Press.

LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. (2019). Graph normalizing flows. *CoRR*, abs/1905.13177.

Nielsen, F. (2010). A family of statistical symmetric divergences based on Jensen's inequality. *arXiv preprint arXiv:1009.4004*.

Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*.

Paige, B. and Wood, F. D. (2016). Inference networks for sequential Monte Carlo in graphical models. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 3040–3049.

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1530–1538.

Ritchie, D., Horsfall, P., and Goodman, N. D. (2016). Deep amortized inference for probabilistic programs. *arXiv:1610.05735 [cs, stat]*. arXiv: 1610.05735.

Ruthotto, L. and Haber, E. (2019). Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13.

Tolpin, D., van de Meent, J.-W., Yang, H., and Wood, F. (2016). Design and implementation of probabilistic programming language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional programming Languages*, pages 1–12.

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

van de Meent, J.-W., Paige, B., Yang, H., and Wood, F. (2018). An introduction to probabilistic programming.

Webb, S., Golinski, A., Zinkov, R., Narayanaswamy, S., Rainforth, T., Teh, Y. W., and Wood, F. (2018). Faithful inversion of generative models for effective amortized inference. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 3074–3084.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2019). A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596.