Flow Models for Arbitrary Conditional Likelihoods

Yang Li [1], **Shoaib Akbar** [2], **Junier B. Oliva** [1]

[1] Department of Computer Science, UNC-Chapel Hill, USA
[2] Department of Mathematics, North Carolina State University, USA
yangli95@cs.unc.edu, sakbar@ncsu.edu, joliva@cs.unc.edu

Abstract

Understanding the dependencies among features of a dataset is at the core of most unsupervised learning tasks. However, a majority of generative modeling approaches are focused solely on the joint distribution $p(x)$ and utilize models where it is intractable to obtain the conditional distribution of some arbitrary subset of features x_u given the rest of the observed covariates x_o: $p(x_u \mid x_o)$. Traditional conditional approaches provide a model for a *fixed* set of covariates conditioned on another *fixed* set of covariates. Instead, in this work we develop a model that is capable of yielding *all* conditional distributions $p(x_u \mid x_o)$ (for arbitrary x_u) via tractable conditional likelihoods. We propose a novel extension of (change of variables based) flow generative models, arbitrary conditioning flow models (AC-Flow), that can be conditioned on arbitrary subsets of observed covariates, which was previously infeasible. We apply AC-Flow to the imputation of features, and also develop a unified platform for both multiple and single imputation by introducing an auxiliary objective that provides a principled single "best guess" for flow models. Extensive empirical evaluations show that our models achieve state-of-the-art performance in both single and multiple imputation across image inpainting and feature imputation in synthetic and real-world datasets. Code is available at https://github.com/lupalab/ACFlow.

1 Introduction

Spurred on by recent impressive results, there has been a surge in interest for generative probabilistic modeling in machine learning. These models learn an approximation of the underlying data distribution and are capable of drawing realistic samples from it. Generative models have a multitude of potential applications, including image restoration (Ledig et al. 2017), agent planning (Houthooft et al. 2016), and unsupervised representation learning (Chen et al. 2016). However, most generative approaches are solely focused on the joint distribution of features, $p(x)$, and are opaque in the conditional dependencies that are carried among subsets of features. In this work, we propose a framework, arbitrary conditioning flow models (AC-Flow), to construct generative models that yield tractable (analytically available) con-

ditional likelihoods $p(x_u \mid x_o)$ of an arbitrary subset of covariates, x_u, given the remaining observed covariates x_o. We focus on the use of AC-Flow for the purpose of imputation, where we infer possible values of x_u, given observed values x_o both in general real-valued data and images (for inpainting).

Classic methods for imputation include k-nearest neighbors (Troyanskaya et al. 2001), random forest (Stekhoven and Bühlmann 2011) and auto-encoder (Gondara and Wang 2018) approaches. Often, these methods infer just one single prediction for each missing value, a.k.a. *single imputation*, and do not provide a conditional distribution. For more complex data, such as high dimensional data or multi-modal data, *multiple imputation* is a more appropriate strategy. This is not only because of the potential to cover multiple modes, but also the ability to quantify the uncertainty of imputations, which may be essential for downstream tasks.

From the perspective of probabilistic modeling, data imputation attempts to learn a distribution of the unobserved covariates, x_u, given the observed covariates, x_o. Thus, generative modeling is a natural fit for data imputation. It handles single imputation and multiple imputation in an unified framework by allowing the generation of an arbitrary number of samples. More importantly, it quantifies the uncertainty in a principled manner. In this work, we combine flow-based and autoregressive approaches for flexible, tractable generative modeling and extend it for data imputation.

Tractable deep generative models are designed to learn the underlying distribution $p(x)$. There are also works that try to model the conditional distribution conditioned on either class labels (Kingma and Dhariwal 2018) or other data points (Li, Gao, and Oliva 2019). For data imputation, however, we are trying to model a subset of the features, x_u conditioned on the rest, x_o, i.e. $p(x_u \mid x_o)$. Although the complete data x are from a certain distribution, the conditional distributions $p(x_u \mid x_o)$ vary when conditioned on different x_o. Furthermore, the dimensionality of x_u and x_o could be arbitrary, for example, when the data are missing completely at random (MCAR, missing independently of covariates' values).

Dealing with arbitrary dimensionality is a rarely explored topic in current generative models. One might want to ex-

plicitly learn a separate model for each different subset of observed covariates. However, this approach quickly becomes infeasible as it requires an exponential number of models with respect to the dimensionality of the input space. In this work, we propose several conditional transformations that handle arbitrary dimensionality in a principled manner.

Our contributions are as follows. 1) We propose a novel extension of flow-based generative models to model the conditional distribution of arbitrary unobserved covariates in data imputation tasks. Our method is the first to develop invertible transformations that operate on an arbitrary set of covariates. 2) We strengthen a flow-based model by using a novel autoregressive conditional likelihood. 3) We propose a novel penalty to generate a single imputed "best guess" for models without an analytically available mean. 4) We run extensive empirical studies and show that AC-Flow achieves state-of-the-art performance for both missing feature imputation and image inpainting on benchmark real-world datasets.

2 Problem Formulation

Consider a real-valued[1] distribution $p(x)$ over $\mathbb{R}^d$. We are interested in estimating the conditional distribution of *all* possible subsets of covariates $u \subseteq \{1, \ldots, d\}$ conditioned on the remaining observed covariates $o = \{1, \ldots, d\} \setminus u$. That is, we shall estimate $p(x_u \mid x_o)$ where $x_u \in \mathbb{R}^{|u|}$ and $x_o \in \mathbb{R}^{|o|}$, for all possible subsets u.

For ease of notation, let $b \in \{0, 1\}^d$ be a binary mask indicating which dimensions are observed. Furthermore, let $v[b]$ index dimensions v_i for which $b_i = 1$ for the bitmask b on a vector v. Thus, $x_o = x[b]$ denotes observed dimensions and $x_u = x[1 - b]$ denotes unobserved dimensions. We also apply this index to matrices such that $W[b, b]$ indexes rows and then columns. Without loss of generality, conditionals may also be conditioned on the bitmask b, $p(x_u \mid x_o, b)$, and will be estimated with maximum log-likelihood estimation as described below. In addition, imputation tasks shall be accomplished by generating samples from the conditional distributions $p(x_u \mid x_o, b)$.

3 Background

AC-Flow builds on Transformation Autoregressive Networks (TANs) (Oliva et al. 2018), a flow-based model that combines transformation of variables with autoregressive likelihoods. We expound on flow-based models and TANs below.

The change of variable theorem (1) is the cornerstone of flow-based generative models, where q represents an invertible transformation that transforms covariates from input space $\mathcal{X}$ into a latent space $\mathcal{Z}$.

$$p_{\mathcal{X}}(x) = \left| \det \frac{dq}{dx} \right| p_{\mathcal{Z}}(q(x)) \quad (1)$$

In order to efficiently compute the determinant, the transformation q is often designed to have a diagonal or triangu-

---
[1] Data with categorical dimensions can be handled by a special case of our model, please refer to appendix A.

lar Jacobian (Dinh, Krueger, and Bengio 2014; Dinh, Sohl-Dickstein, and Bengio 2016; Kingma and Dhariwal 2018). Since this type of transformation is rather restrictive, the flow models are often composed of multiple transformations in a sequence to get a more flexible composite transformation, i.e. $q = q_m \circ q_{m-1} \circ \ldots \circ q_1$. Here, the covariates flow through a chain of transformations, substituting the last output variable as input for the next transformation.

Typically, a flow-based model transforms the covariates to a latent space with a simple base distribution, like a standard Gaussian. However, TANs provides more flexibility by modeling the latent distribution with an autoregressive approach (Larochelle and Murray 2011). This alters the earlier equation (1), in that $p_{\mathcal{Z}}(q(x))$ is now represented as the product of d conditional distributions.

$$p_{\mathcal{X}}(x) = \left| \det \frac{dq}{dx} \right| \prod_{i=1}^{d} p_{\mathcal{Z}}(z^i \mid z^{i-1}, \ldots, z^1) \quad (2)$$

Since flow models give the exact likelihood, they can be trained by directly optimizing the log likelihood. In addition, thanks to the invertibility of the transformations, one can draw samples by simply inverting the transformations over a set of samples from the latent space.

4 Methods

We now develop AC-Flow by constructing both conditional transformations of variables and autoregressive likelihoods that work with an arbitrary set of unobserved covariates. To deal with arbitrary dimensionality for conditioning covariates x_o, we define a zero imputing function $\phi(v; b)$ that imputes input vector v with zeros based on the specified binary mask b:

$$w = \phi(v; b), \quad w_i = \begin{cases} v_c, & b_i = 1 \\ 0, & b_i = 0 \end{cases}, \quad (3)$$

where $c = \sum_{j=1}^{i} b_j$ represents the cumulative sum over b. For example, $\phi(x_o; b)$ gives a d-dimensional vector with missing values imputed by zeros (See Fig. 1(a) for an illustration.). Thus, we get a conditioning vector with fixed dimensionality. However, handling the arbitrary dimensionality of x_u requires further care, as discussed below.

We first consider a conditional extension to the change of variable theorem:

$$p_{\mathcal{X}}(x_u \mid x_o, b) = \left| \det \frac{dq_{x_o,b}}{dx_u} \right| p_{\mathcal{Z}}(q_{x_o,b}(x_u) \mid x_o, b), \quad (4)$$

where $q_{x_o,b}$ is a transformation on the unobserved covariates x_u with respect to the observed covariates x_o and binary mask b as demonstrated in Fig. 1(a). However, the fact that x_u could have varying dimensionality and arbitrary missing dimensions makes it challenging to define $q_{x_o,b}$'s across different bitmasks b. One challenge comes from requiring the transformation to have adaptive outputs that can adapt to different dimensionality of x_u. Another challenge is that different missing patterns require the transformation to capture different dependencies. Since the missing pattern could be arbitrary, we require the transformation to learn a large
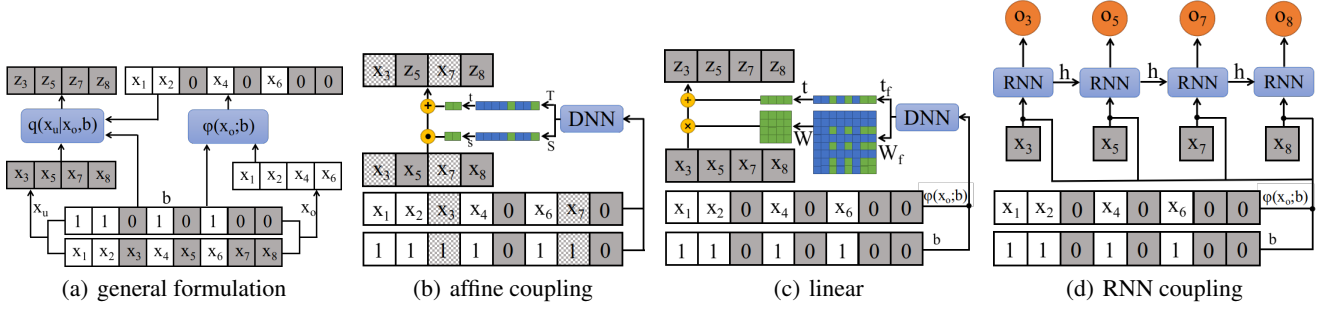
Figure 1: Conditional transformations used in AC-Flow. Grayed out boxes represent missing covariates. Checkerboarded boxes in (b) belong to unobserved dimensions, but are used as conditioning in the affine coupling transformation.

range of possible dependencies. Aiming at solving those challenges, we propose several conditional transformations that leverage the conditioning information in x_o and b and can be adapted to x_u with arbitrary dimensionality. We describe them in detail below. For notation simplicity, we drop the subscripts $\mathcal{X}$ and $\mathcal{Z}$ in the following sections.

Affine Coupling Transformation Affine coupling is a commonly used flow transformation. Here, we derive a conditional extension to it. Just as in the unconditional counterparts (Dinh, Krueger, and Bengio 2014; Dinh, Sohl-Dickstein, and Bengio 2016; Kingma and Dhariwal 2018), we divide the unobserved covariates x_u into two parts, x_u^A and x_u^B according to some binary mask $b_u \in \{0,1\}^{|u|}$, i.e., $x_u^A = x_u[b_u]$ and $x_u^B = x_u[1-b_u]$. We then keep the first part x_u^A and transform the second part x_u^A, i.e.,

$$
\begin{aligned}
z_u^A &= x_u^A \\
z_u^B &= x_u^B \odot s(x_u^A, b_u, x_o, b) + t(x_u^A, b_u, x_o, b),
\end{aligned} \tag{5}
$$

where $\odot$ represents the element-wise (Hadamard) product. s and t computes the scale and shift factors as the function of both observed covariates and covariates in group A. Note that both inputs and outputs of s and t contain arbitrary dimensional vectors. To deal with arbitrary dimensionality in inputs, we apply the zero imputing function (3) to x_u^A and x_o respectively to get two d-dimensional vectors with missing values imputed by zeros. We also apply ϕ to b_u to get a d-dimensional mask. The shift and scale functions s and t are implemented as deep neural networks (DNN) over $x_c = \phi(\phi(x_u^A; b_u); 1-b) + \phi(x_o; b)$ and $b_c = \phi(b_u; 1-b) + b$, i.e,

$$
\begin{aligned}
S &= \mathrm{DNN}(\mathbf{concat}(x_c, b_c)) \in \mathbb{R}^d, \\
T &= \mathrm{DNN}(\mathbf{concat}(x_c, b_c)) \in \mathbb{R}^d,
\end{aligned} \tag{6}
$$

where **concat** defines a concatenate function and the two DNN functions can share weights.

The outputs of s and t need to be adaptive to the dimensions of x_u^B, thus we apply the indexing mechanism, $[\cdot]$, that takes the corresponding dimensions of non-zero values with respect to binary masks $1-b$ and $1-b_u$, i.e.,

$$
\begin{aligned}
s &= S\,[1-b]\,[1-b_u], \\
t &= T\,[1-b]\,[1-b_u].
\end{aligned} \tag{7}
$$

A visualization of this transformation is presented in Fig. 1(b).

Note that the way we divide x_u might depend on our prior knowledge about the data correlations. For image data, we use checkerboard and channel-wise split as in (Dinh, Sohl-Dickstein, and Bengio 2016). For real-valued vectors, we use even-odd split as in (Dinh, Krueger, and Bengio 2014).

Linear Transformation Another common transformation for flow-based models is the linear transformation. Contrary to the coupling transformation that only leverages correlation between two separated subsets, the linear transformation can take advantage of correlation between all dimensions. Furthermore, the linear transformation can be viewed as a generalized permutation which rearranges dimensions so that next transformation can be more effective.

In order to transform x_u linearly, we would like to have an adaptive weight matrix $\mathbf{W}$ of size $|u| \times |u|$ and a bias vector t of size $|u|$. Similar to the affine coupling described above, we first apply a deep neural network over $\phi(x_o; b)$ and binary mask b to get a $d \times d$ matrix $\mathbf{W_f}$ and a d-dimensional bias vector t_f, then we index them with respect to the binary mask $1-b$, i.e.,

$$
\begin{aligned}
\mathbf{W_f} &= \mathrm{DNN}(\phi(x_o; b), b) \in \mathbb{R}^{d \times d}, \\
\mathbf{W} &= \mathbf{W_f}\,[1-b, 1-b] \in \mathbb{R}^{|u| \times |u|}, \\
t_f &= \mathrm{DNN}(\phi(x_o; b), b) \in \mathbb{R}^d, \\
t &= t_f\,[1-b] \in \mathbb{R}^{|u|}.
\end{aligned} \tag{8}
$$

The linear transformation can then be derived as $z_u = \mathbf{W}x_u + t$. Fig. 1(c) illustrates this transformation over an 8-dimensional example.

In order to decrease complexity, it is straightforward to parametrize W_f with rank r matrices by taking the product of two rank r matrices with size $d \times r$ and $r \times d$ respectively. Hence, the DNN can reduce its output dimensions to $2dr$. During preliminary experiments, we observed minimal drop in performance when using a large enough r.

RNN Coupling Transformation The affine coupling transformation can be viewed as a rather rough recurrent transformation with only one recurrent step. We can generalize it by running an RNN over x_u and transform each dimension sequentially (shown in Fig.1(d)). Note that a recur-

rent transformation naturally handles different dimensionality. To leverage conditioning inputs x_o and b, we concatenate $\phi(x_o; b)$ and b to each dimension of x_u. The outputs of the RNN are used to derive the shift and scale parameters respectively.

$$o^i, h^i = \text{RNN}(\textbf{concat}(x_u^{i-1}, \phi(x_o; b), b), h^{i-1}),$$
$$z_u^i = x_u^i * s(o^i) + t(o^i), \tag{9}$$

where $x_u^0 = -\mathbf{1}$, $h^0 = \mathbf{0}$, and i indexes through dimensions.

Other Transformations and Compositions Other transformations like element-wise leaky-ReLU transformation (Oliva et al. 2018) and reverse transformation (Dinh, Krueger, and Bengio 2014) are readily applicable to transform the unobserved covariates x_u since they do not rely on the conditioning covariates. Other than these specific transformations described above, any transformations that follow the general formulation shown in Fig. 1(a) can be easily plugged into our model. We obtain flexible, highly non-linear transformations with the composition of multiple of these aforementioned transformations. (That is, we use the output of the preceding transformation as input to the next transformation.) The Jacobian of the resulting composed (stacked) transformation is accounted with the chain rule.

Likelihoods The conditional likelihoods in latent space $p(z_u | x_o, b)$ can be computed by either a base distribution, like a Gaussian, or an autoregressive model as in TANs. For Gaussian based likelihoods, we can get mean and covariance parameters by applying another function over $\phi(x_o; b)$ and b, which is essentially equivalent to another linear transformation conditioned on x_o and b. However, this approach is generally less flexible than using an autoregressive approach.

For autoregressive likelihoods, conditioning vectors can be used in the same way as the RNN coupling transformation. The difference is that the RNN outputs are now used to derive the parameters for some base distribution, for example, a Gaussian Mixture Model:

$$o^i, h^i = \text{RNN}(\textbf{concat}(z_u^{i-1}, \phi(x_o; b), b), h^{i-1}),$$
$$p(z_u^i | z_u^{i-1}, ..., z_u^1, x_o, b) = \text{GMM}(z_u^i | \theta(o^i)), \tag{10}$$

where $\theta(o^i)$ is a shared fully connected network that maps to the parameters for the mixture model (i.e. each mixture component's location, scale, and prior weight parameter). During sampling, we iteratively sample each point, z_u^{i-1}, before computing the parameters for z_u^i. Incorporating the autoregressive likelihood into Eq. (4) yields:

$$p(x_u | x_o, b) = \left| \det \frac{dq_{x_o,b}}{dx_u} \right| \prod_{i=1}^{|u|} p(z_u^i | z_u^{i-1}, ..., z_u^1, x_o, b), \tag{11}$$

where $|u|$ is the cardinality of the unobserved covariates.

4.1 Training and Best Guess Objective

Since we follow a flow-based approach, we have access to the normalized conditional likelihoods; thus, we can train our model by maximizing log likelihood.

During training, if we have access to complete training data, we will need to manually create binary masks b based on some predefined distribution p_b. p_b is typically chosen based on the application. For instance, Bernoulli random masks are commonly used for real-valued vectors. Given binary masks, training data x are divided into x_u and x_o and fed into the conditional model $p(x_u | x_o, b)$.

If training data already contains missing values, we can only train our model on the remaining covariates. As before, we manually split each data point into two parts, x_u and x_o based on a binary mask b. Note that dimensions in b corresponding to the missing values are always set to 0, i.e., they are never observed during training. In this setting, we will need another binary mask m indicating those dimensions that are not missing. Accordingly, we define observed dimensions as $x_o = x[b]$ and unobserved dimensions as $x_u = x[m \odot (1 - b)]$. In this setting, we optimize $p(x_u | x_o, m, b)$. During testing, we set $b = m$, that is, the model imputes the missing values conditioned on all the remaining covariates.

Best Guess Objective Given a trained AC-Flow model, multiple imputations can be easily accomplished by drawing multiple samples from the learned conditional distribution. However, certain downstream tasks may require a single imputed "best guess". Unfortunately, the analytical mean $\mathbb{E}_{p(x_u|x_o,b)}[x_u]$ is not available for flow-based deep generative models. Furthermore, getting an accurate empirical estimate could be prohibitive in high dimensional space. In this work, we propose a robust solution that gives a single best guess in terms of the MSE metric (it can be easily extended to other metrics, e.g. an adversarial one).

Specifically, we obtain our best guess by inverting the conditional transformation over the mean of the latent distribution, i.e., $q_{x_o,b}^{-1}(\bar{z}) = q_{x_o,b}^{-1}(\mathbb{E}_{p_\mathcal{Z}(z|x_o,b)}[z])$. The mean $\bar{z}$ is analytically available for Gaussian mixture base model. To ensure that this best guess is close to unobserved values, we optimize with an auxiliary MSE loss:

$$\mathcal{L} = -\log\ p(x_u | x_o, b) + \lambda \| q_{x_o,b}^{-1}(\bar{z}) - x_u \|^2, \tag{12}$$

where λ controls the relative importance of the auxiliary objective. Note that we only penalize one particular point from $p(x_u | x_o, b)$ to be close to x_u. Hence, it does not affect the diversity of the conditional distribution.

5 Related Work

5.1 Conditional Generative Models

Unlike our arbitrary conditioning task, conditional generative modeling deals only with a fixed vector of external information y (such as a class label) to condition the likelihood of a fixed set of covariates x (such as pixels in an image). Typically, approaches supplement the inputs of common generative models with some encoding of the conditional information. Conditional GANs (Mirza and Osindero 2014), for example, extended GANs by inputting a class label encoding into both the generator and discriminator function, allowing these models to learn multiple conditional distributions. Similarly, a conditional form of VAE was proposed by
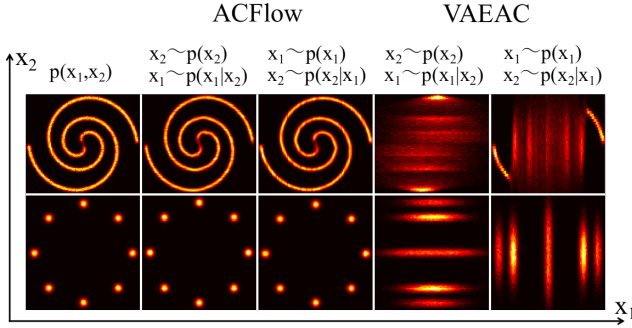
Figure 2: Synthetic data imputation. Observed covariates are sampled from the marginal distributions, and the missing covariates are sampled from the learned conditional distributions.

Table 1: Image inpainting results. Mean and std. dev. reported on 5 random binary masks for each testing data. For PSNR and PRD, higher is better. For NLL, lower is better.

		MNIST	Omniglot	CelebA
VAEAC	PSNR	19.613±0.042	17.693±0.023	23.656±0.027
	NLL	3835.977±6.048	4015.213±2.530	56300.842±15.214
	PRD	(0.975, 0.877)	(0.926, 0.525)	(0.966, 0.967)
AC-Flow	PSNR	17.349±0.018	15.572±0.031	22.393±0.040
	NLL	1919.555±2.500	1834.552±0.695	**39376.466±47.670**
	PRD	(**0.984**, 0.945)	(**0.971**, 0.962)	(**0.988**, **0.970**)
AC-Flow BG	PSNR	**20.828±0.031**	**18.838±0.009**	**25.723±0.020**
	NLL	**1916.887±1.190**	**1833.068±0.760**	39504.883±20.475
	PRD	(0.983, **0.947**)	(0.970, **0.967**)	(0.987,0.964)

(Sohn, Yan, and Lee 2015), which introduces an additional network that learns the conditional prior $p_\theta(z \mid y)$, where z is some latent encoding and y is a class label.

5.2 Arbitrary Conditioning Models

Previous attempts to learn probability distributions conditioned on arbitrary subsets of known covariates include the Universal Marginalizer (Douglas et al. 2017), which is trained as a feed-forward network to approximate the marginal posterior distribution of each unobserved dimension conditioned on the observed ones. During sampling, they propose to use a sequential sampling mechanism by adding a newly sampled dimension to the observed sets and running the network in an autoregressive manner. GAIN approaches data imputation with a GAN approach (Yoon, Jordon, and Schaar 2016). The model uses a generator that provides imputations for missing data points while the discriminator attempts to distinguish imputed covariates from observed covariates. VAEAC (Ivanov, Figurnov, and Vetrov 2019) utilizes a conditional variational autoencoder and extends it to deal with arbitrary conditioning. The decoder network outputs likelihoods that are over all possible dimensions, although, since they are conditionally independent given the code, it is possible to use only the likelihoods corresponding to the unobserved dimensions. However, we observe that VAEAC suffers from failure modes common in VAEs given that they optimize with respect to ELBO loss (Zhao and Ermon 2019). In particular, VAEs sometimes sacrifice learning the true posterior for minimizing reconstruction loss, which often leads the model to generate blurry samples for multimodal distributions.

Unlike VAEAC and GAIN, AC-Flow is capable of producing an analytical (normalized) likelihood and avoids blurry samples and other failure modes in VAE approaches. Furthermore, in contrast to the Universal Marginalizer, AC-Flow models dependencies between unobserved covariates at training time and makes use of change of variables.

6 Experiments

In this section, we evaluate our proposed model against VAEAC (Ivanov, Figurnov, and Vetrov 2019), GAIN (Yoon, Jordon, and Schaar 2016) and other common imputation methods such as MICE (Buuren and Groothuis-Oudshoorn 2010) and MissForest (Stekhoven and Bhlmann 2012). We compare these models on both missing feature imputation tasks and image inpainting tasks below.

6.1 Synthetic Data Imputation

To validate the effectiveness of our model, we conduct experiments on synthetic 2-dimensional datasets (Behrmann, Duvenaud, and Jacobsen 2018), i.e., $x = (x_1, x_2) \in \mathbb{R}^2$. The joint distributions are plotted in Fig. 2. Here, the conditional distributions are highly multi-modal and vary significantly conditioned on different observations. We compare our model against VAEAC by training them on 100,000 samples from the joint distribution. The masks are generated by dropping out one dimension at random. The details about training procedure are provided in Appendix B. Fig. 2 shows imputation results form our model and VAEAC. We show imputations from both $p(x_1 \mid x_2, b)$ and $p(x_2 \mid x_1, b)$ by generating 10 samples conditioned on each observed covariate. Observed cavariates are sampled from the marginal distribution $p(x_2)$ and $p(x_1)$ respectively. We see that AC-Flow is capable of learning multi-modal distributions, while VAEAC tends to merge multiple modes into one single mode.

6.2 Image Inpainting

In this section, we evaluate our method on image inpainting tasks using three common datasets, MNIST, Omniglot and CelebA with a varied distribution of binary masks. Details about mask generation and image preprocessing are available in the Appendix C.1.

We extend the RealNVP model (Dinh, Sohl-Dickstein, and Bengio 2016) by replacing all the coupling layers with our proposed arbitrary conditional alternative. For the sake

(a) MNIST inpaintings

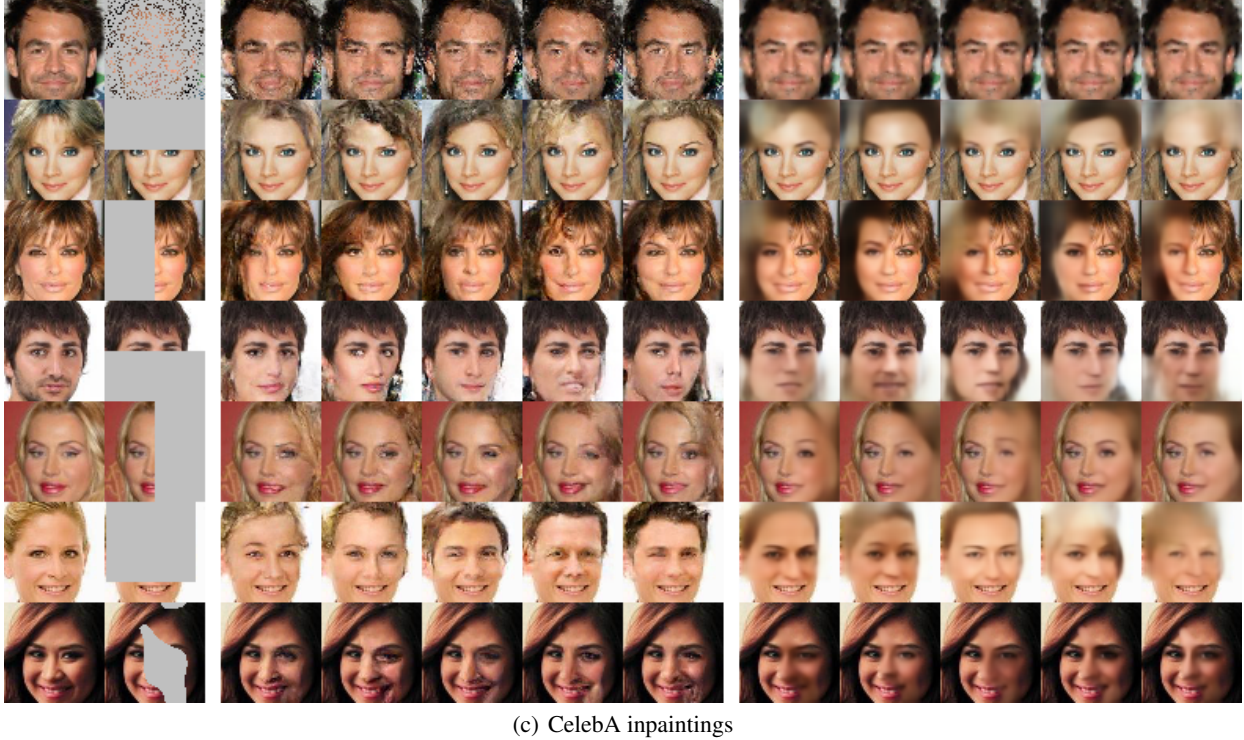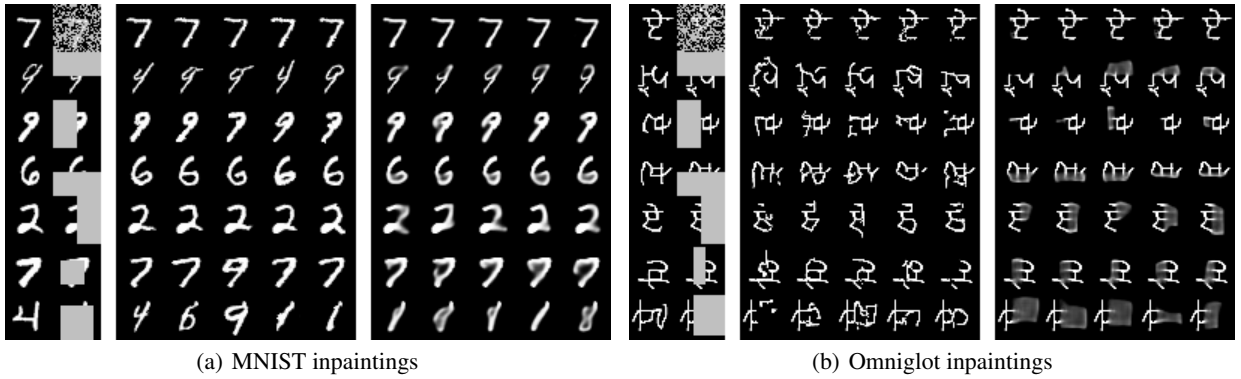(b) Omniglot inpaintings

(c) CelebA inpaintings

Figure 3: Image Inpaintings. Left: groundtruth and inputs. Middle: samples from AC-Flow. Right: samples from VAEAC.

of sampling efficiency, we use standard Gaussian base likelihoods here. Implementation details of AC-Flow and baselines are provided in Appendix C.2.

Figure. 3 shows samples drawn from AC-Flow and VAEAC. More samples are available in Appendix C.3. We notice our model can generate coherent and diverse inpaintings for all three datasets and different masks. Compared to VAEAC, our model is capable of generating sharp samples and restoring more details. Even when the missing rate is high, AC-Flow can still generate decent inpaintings.

To quantitatively evaluate our model, we report peak signal-to-noise ratio (PSNR) and negative log-likelihoods (NLL) in Table. 1. We generate 5 different masks for each test image and report the average scores and the standard deviation. We note that PSNR is a metric that prefers blurry images over sample diversity (Hore and Ziou 2010). Log-

likelihood measures how well the model matches the real conditional distribution, but may not correlate with visual quality (Theis, Oord, and Matthias 2016). Hence, we evaluate the trade-off between sample quality and diversity via the precision and recall scores (PRD) (Sajjadi et al. 2018). Since we cannot sample from the groundtruth conditional distribution, we compute the PRD score between the imputed joint distribution $p(x_o)p(x_u \mid x_o)$ and the true joint distribution $p(x)$ via 10,000 samples from them. The PRD scores for two distributions measure how much of one distribution can be generated by another. Higher recall means a greater portion of samples from the true distribution $p(x)$ are covered by $p(x_o)p(x_u \mid x_o)$; and similarly, higher precision means a greater portion of samples from $p(x_o)p(x_u \mid x_o)$ are covered by $p(x)$. We report the $(F_8, F_{\frac{1}{8}})$ pairs in Table. 1 to represent recall and precision, respectively.

(a) Normalized Root Mean Square Error (NRMSE)

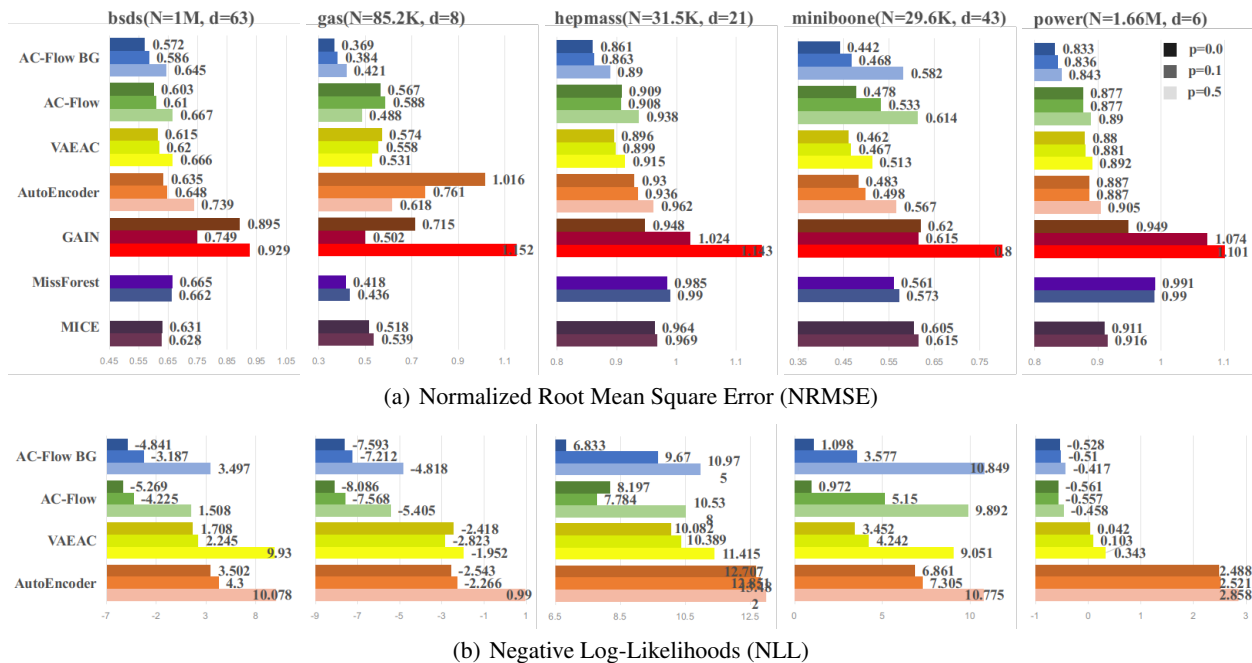(b) Negative Log-Likelihoods (NLL)

Figure 4: NRMSE and NLL results for real-valued feature imputation on UCI datasets. Lower is better. On top of each chart, we present the dataset name and its training set size (N) and feature dimensions (d). For each method, dark color represents results from a model trained with complete training data; medium color represents missing rate of 0.1; light color means missing rate of 0.5. We test MICE and MissForest only when the missing rate is not zero. For MICE, MissForest and GAIN, we do not have access to the exact likelihoods. "AC-Flow BG" represents our model trained with the "Best Guess" penalty.

From quantitative results, we can see that our model gives lower NLL and higher PRD, which means our model does better in learning the true distribution. Training with the auxiliary "best guess" penalty (denoted as "AC-Flow BG") can improve the PSNR scores significantly, but hardly impacts the likelihood and PRD scores. Thus, our proposed penalty provides a "free lunch" for the image inpainting task.

6.3 Real-valued Feature Imputation

Next, we evaluate our model on missing feature imputation. We use UCI repository datasets preprocessed as described in (Papamakarios, Pavlakou, and Murray 2017). We construct models by composing several leaky-ReLU, conditional linear, and RNN coupling transformations, along with an autoregressive conditional likelihood (see Appendix. D.1 for further architectural details).

First, we consider fully observed training data, where missing covariates are to be imputed at test time. We construct masks, b, by dropping a random subset of the dimensions according to a Bernoulli distribution with $p = 0.5$. Afterwards, we also evaluate our model when the data itself contains missing values that are never available during training. We consider training and testing with data features missing completely at random at a 10% and 50% rate.

Baseline models for this experiment include MICE, Miss-Forest, GAIN and VAEAC. We also compare to a denoising autoencoder in which missing covariates are treated as dropout noise. Implementation details about AC-Flow and

baselines are provided in Appendix D.1.

In Fig. 4, we show the NRMSE (i.e. root mean squared error normalized by the standard deviation of each feature and then averaged across all features) and the test NLL (if available) for AC-Flow and baselines. For models that can perform multiple imputation, 10 imputations are drawn and averaged for each test point to compute the NRMSE scores. For our model trained with the auxiliary objective, we use the single "best guess" to compute the NRMSE. In order to not bias towards any specific missing pattern, we report the mean and standard deviations over 5 randomly generated binary masks (std. dev. are reported in Appendix, Table. D.1).

One can see that our model gives the best NLL on nearly all scenarios, which indicates our model is better at learning the true distribution. In terms of the NRMSE, AC-Flow is comparable to the previous state-of-the-art when trained purely by maximizing the likelihood. However, training with the auxiliary objective (AC-Flow BG) improves the NRMSE scores significantly and gives state-of-the-art results on all datasets considered. As expected, training with missing values is more difficult; however, our model performs best even when the missing rate is relatively high.

7 Conclusion

In this work, we demonstrated that we can improve samples generated with arbitrary conditioning on NRMSE, PSNR, log-likelihood, visual quality, and PRD by leveraging conditional flow-models and conditional autoregressive likeli-

hoods. We also considered the task of training both single and multiple imputations in a unified platform to provide a "best guess" single imputation when the mean is not analytically available. The samples generated from our model show that we improve in both diversity and quality of imputations in many datasets. Our model typically recovers more details than the previous state-of-the-art methods.

References

Behrmann, J.; Duvenaud, D.; and Jacobsen, J.-H. 2018. Invertible residual networks. *arXiv preprint arXiv:1811.00995*.

Buuren, S. v., and Groothuis-Oudshoorn, K. 2010. Mice: Multivariate imputation by chained equations in r. In *Journal of statistical software*, 1–68.

Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; and Abbeel, P. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, 2172–2180.

Dinh, L.; Krueger, D.; and Bengio, Y. 2014. Nice: Non-linear 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*.

Douglas, L.; Zarov, I.; Gourgoulias, K.; Lucas, C.; Hart, C.; Baker, A.; Sahani, M.; Perov, Y.; and Johri, S. 2017. A universal marginalizer for amortized inference in generative models. *arXiv preprint arXiv:1711.00695*.

Gondara, L., and Wang, K. 2018. Mida: Multiple imputation using denoising autoencoders. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 260–272. Springer.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hore, A., and Ziou, D. 2010. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, 2366–2369.

Houthooft, R.; Chen, X.; Duan, Y.; Schulman, J.; De Turck, F.; and Abbeel, P. 2016. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, 1109–1117.

Ivanov, O.; Figurnov, M.; and Vetrov, D. 2019. Variational autoencoder with arbitrary conditioning. In *International Conference on Learning Representations*.

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

Larochelle, H., and Murray, I. 2011. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 29–37.

Ledig, C.; Theis, L.; Huszár, F.; Caballero, J.; Cunningham, A.; Acosta, A.; Aitken, A.; Tejani, A.; Totz, J.; Wang, Z.; et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4681–4690.

Li, Y.; Liu, S.; Yang, J.; and Yang, M.-H. 2017. Generative face completion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3911–3919.

Li, Y.; Gao, T.; and Oliva, J. 2019. A forest from the trees: Generation through neighborhoods. *arXiv preprint arXiv:1902.01435*.

Mirza, M., and Osindero, S. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Oliva, J. B.; Dubey, A.; Zaheer, M.; Póczos, B.; Salakhutdinov, R.; Xing, E. P.; and Schneider, J. 2018. Transformation autoregressive networks. *arXiv preprint arXiv:1801.09819*.

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

Pathak, D.; Krahenbuhl, P.; Donahue, J.; Darrell, T.; and Efros, A. A. 2016. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2536–2544.

Sajjadi, M. S.; Bachem, O.; Lucic, M.; Bousquet, O.; and Gelly, S. 2018. Assessing generative models via precision and recall. In *Advances in Neural Information Processing Systems*, 5228–5237.

Sohn, K.; Yan, X.; and Lee, H. 2015. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, 3483–3491.

Stekhoven, D. J., and Bühlmann, P. 2011. Missforestnonparametric missing value imputation for mixed-type data. *Bioinformatics* 28(1):112–118.

Stekhoven, D. J., and Bhlmann, P. 2012. Missforest non-parametric missing value imputation for mixed-type data. In *Bioinformatics, Volume 28, Issue 1*, 112–118.

Theis, L.; Oord, A. v. d.; and Matthias, B. 2016. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

Troyanskaya, O.; Cantor, M.; Sherlock, G.; Brown, P.; Hastie, T.; Tibshirani, R.; Botstein, D.; and Altman, R. B. 2001. Missing value estimation methods for dna microarrays. *Bioinformatics* 17(6):520–525.

Yoon, J.; Jordon, J.; and Schaar, M. v. d. 2016. Gain: Missing data imputation using generative adversarial nets. *arXiv preprint arXiv:1605.08803*.

Zhao, Shengjia, S. J., and Ermon, S. 2019. Infovae: Information maximizing variational autoencoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 5885–5892.

Appendices

A AC-Flow for Categorical Data

In the main text, we mainly focus on real-valued data, but our model is also applicable with data that contains categorical dimensions. Consider a d-dimensional vector x with both real-valued and categorical features. The conditional distribution $p(x_u|x_o)$ can be factorized as $p(x_u^c|x_o)p(x_u^r|x_o, x_u^c)$, where x_u^r and x_u^c represent the real-valued and categorical components in x_u respectively. Since the conditioning inputs in Eq. (4) can be either real-valued or categorical, $p(x_u^r|x_o, x_u^c)$ can be directly modeled by AC-Flow. However, AC-Flow is not directly applicable to $p(x_u^c|x_o)$, because the change of variable theorem, Eq. (1), cannot be applied to categorical covariates. But we can use an autoregressive model, i.e. AC-Flow without any transformations, to get likelihoods for categorical components. Hence, our proposed method can be applied to both real-valued and categorical data.

B Synthetic Data Imputation

We closely follow the released code for VAEAC[1] to construct the proposal, prior, and generative networks. Specifically, we use fully connected layers and skip connections as is in their official implementation. We also use short-cut connections between the prior network and the generative network. We search over different combinations of the number of layers, the number of hidden units of fully connected layers, and the dimension of the latent code. Validation likelihoods are used to select the best model.

AC-Flow is constructed by stacking multiple conditional transformations and an autoregressive likelihood. One conditional transformation layer (shown in Fig. B.1) contains one linear transformation, followed by one leaky relu transformation, and then one RNN coupling transformation. The DNN in the linear transformation is a 2-layer fully connected network with 256 units. RNN coupling is implemented as a 2-layer GRU with 256 hidden units. We stack four conditional transformation layers and use reverse transformations in between. The autoregressive likelihood model is a 2-layer GRU with 256 hidden units. The base distribution is a Gaussian Mixture Model with 40 components.

C Image Inpainting

C.1 Datasets and Masks

We use MNIST, Omniglot, and CelebA to evaluate our model on image inpainting tasks. MNIST contains grayscale images of size 28×28. Omniglot data is also resized to 28×28 and augmented with rotations. For CelebA dataset, we take a central crop of 128×128 and then resize it to 64×64. Since a flow model requires continuous inputs, we dequantize the images by adding independent uniform noise to them.

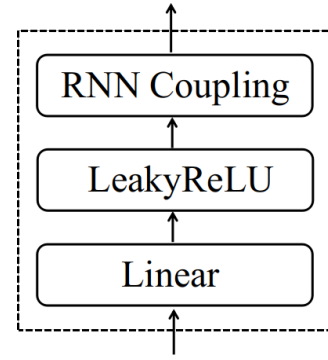[1]https://github.com/tigvarts/vaeac/blob/master/imputation_networks.py

Figure B.1: One conditional transformation layer.

In order to obtain inpaintings for different masks, we train AC-Flow and VAEAC using a mixture of different masks. For MNIST and Omniglot, we use MCAR masks, rectangular masks, and half masks. For CelebA, other than the three masks described above, we also use a random pattern mask proposed in (Pathak et al. 2016) and the masks used in (Li et al. 2017). We shall describe the meaning of different masks below.

MCAR mask MCAR mask utilizes a pixelwise independent Bernoulli distribution with $p = 0.2$ to construct masks. That is, we randomly drop out 80% of the pixels.

Rectangular mask We randomly generate a rectangle inside which pixels are marked as unobserved. The mask is generated by sampling a point to be the upper-left corner and randomly generating the height and width of the rectangle, although the area is required to be at least 10% of the full image area.

Half mask Half mask means either the upper, lower, left or right half of the image is unobserved.

Random pattern mask Random pattern means we mask out a random region with an arbitrary shape. We take the implementation from VAEAC[2].

Masks in (Li et al. 2017) They proposed six different masks to mask out different parts of a face, like eyes, mouth and checks.

C.2 Models

For this experiment, AC-Flow is implemented by replacing affine coupling in RealNVP (Dinh, Sohl-Dickstein, and Bengio 2016) with our proposed conditional affine coupling transformation. The DNN in affine coupling is implemented as a ResNet (He et al. 2016). For MNIST and Omniglot, we use 2 residual blocks. For CelebA, we use 4 blocks. We also

---

[2]https://github.com/tigvarts/vaeac/blob/master/mask_generators.py#L100

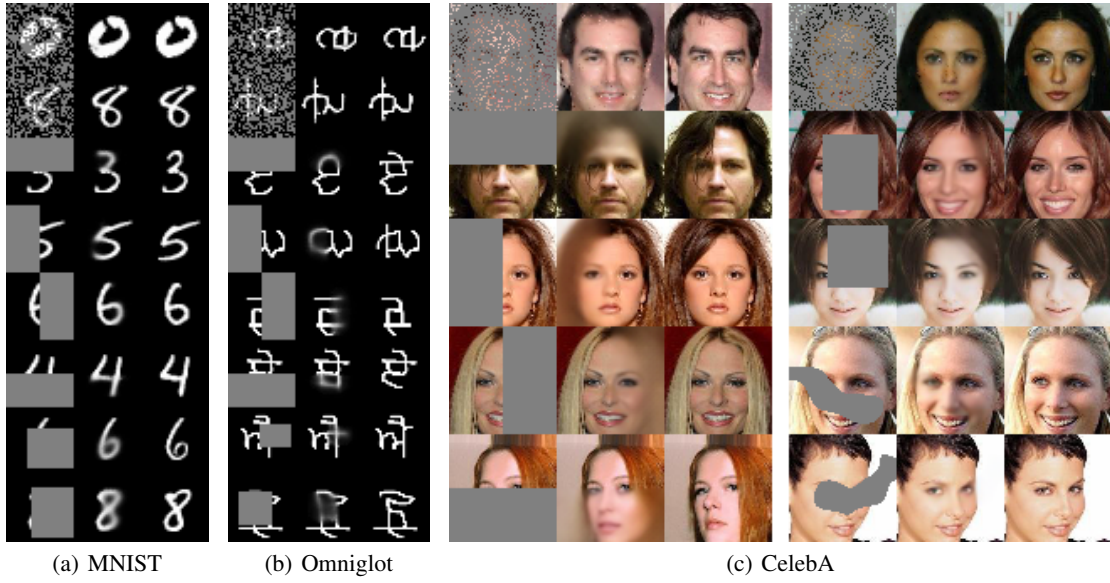| (a) MNIST | (b) Omniglot | (c) CelebA |

Figure C.2: Single imputation from our "best guess". Left: inputs. Middle: best guess imputation. Right: groundtruth.

use the squeeze operation to enable large receptive fields. For MNIST and Omniglot, we use 3 squeeze operations. For CelebA, we use 4 squeeze operations. Note that we also need to squeeze the binary mask b in the same way to make sure it corresponds to x_o.

For AC-Flow BG, we set the hyperparameter λ to 1.0 in all experiments. During preliminary experiments, we find our model is quite robust to different λ values.

VAEAC released their implementation for the 128×128 CelebA dataset and we modify their code by removing the first pooling layer to suit 64×64 images. For MNIST and Omniglot, we build similar networks by using less convolution and pooling layers. Specifically, we pad the images to 32×32 and use 5 pooling layers to get a latent vector. We use 32 latent variables for MNIST and Omniglot and 256 latent variables for CelebA. We also use short-cut connections between the prior network and the generative network as in (Ivanov, Figurnov, and Vetrov 2019).

C.3 Additional Samples

We show additional samples from AC-Flow in Fig. C.3. We also show some "best guess" inpaintings obtained by a model trained with the auxiliary objective in Fig. C.2. As we expected, the "best guess" imputations tend to be blurry due to the MSE penalty. However, the samples from the same model are still diverse and coherent as can be seen from Fig. C.4.

D Real-valued Feature Imputation

D.1 Models

For this experiment, we build AC-Flow by combining 6 conditional transformation layers (shown in Fig. B.1) with a 4-layer GRU autoregerssive likelihood model. The base distribution is a Gaussian Mixture with 40 components. AC-Flow BG uses the same network architecture but trained with

$\lambda = 1.0$. We search over 0.1, 1.0, and 10.0 for λ and find it gives better likelihood for all datasets when $\lambda = 1.0$. Hence, we only report results when $\lambda = 1.0$ in the main text.

We modify the official VAEAC implementation for this experiment. In their original implementation, they restricted the generative network to output a distribution with variance equal to 1. We found that learning the variance can improve VAEAC's likelihood significantly and gives them comparable NRMSE scores. Note that during sampling, we still use the mean of the outputs from the generative network, because it gives slightly better NRMSE scores.

MICE and MissForest are trained using default parameters in the R packages.

For GAIN, we use the variant the VAEAC authors proposed. They observed consistent improvement over the original one on UCI datasets. To apply GAIN when we have complete training data, we add another MSE loss over the unobserved covariates to train the generator.

The autoencoder is implemented as a 6-layer fully connected network with 256 hidden units. ReLU is used after each fully connected layer except the last one. We use standard Gaussian as the base distribution.

D.2 Results

We list all results in Table. D.1. We report mean and standard deviation by generating 5 different masks for each test data point.
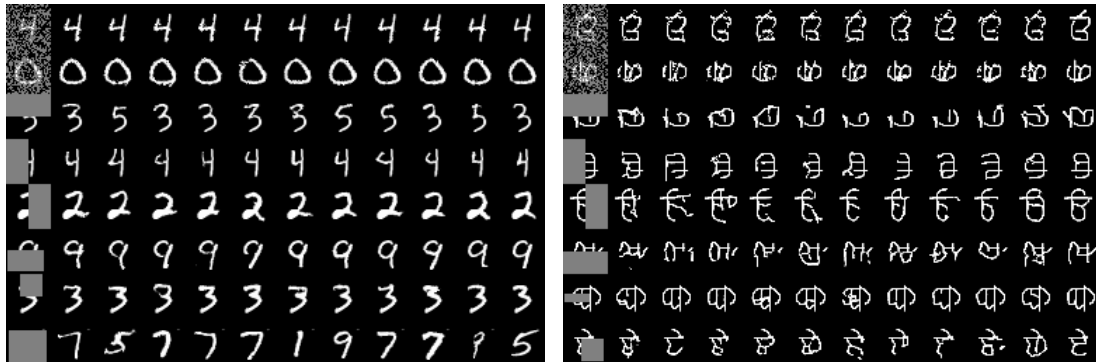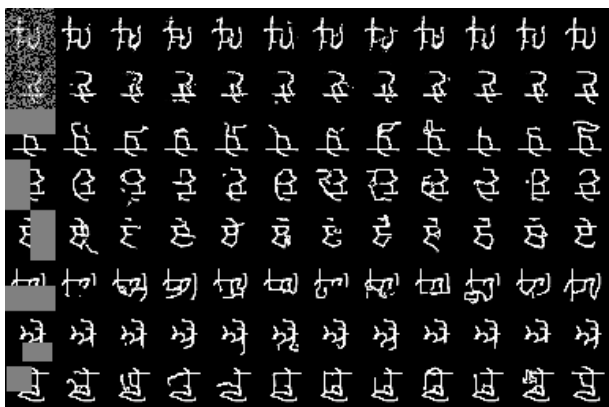
(a) MNIST

(b) Omniglot

(c) CelebA

Figure C.3: Additional samples from AC-Flow. Left:inputs. Middle: Samples. Right: groundtruth.

(a) MNIST

(b) Omniglot

(c) CelebA

Figure C.4: Multiple imputations from AC-Flow BG.Left:inputs. Middle: Samples. Right: groundtruth.

Table D.1: Missing feature imputation results. Lower is better for both NRMSE and NLL. Numbers inside a parentheses are standard deviation for 5 randomly generated binary mask.

Missing Rate	Method		bsds	gas	hepmass	miniboone	power
p=0	GAIN	NRMSE	0.895±0.151	0.715±0.041	0.948±0.006	0.620±0.002	0.949±0.017
	AutoEncoder	NRMSE	0.635±0.000	1.016±0.178	0.930±0.001	0.483±0.003	0.887±0.002
		NLL	3.502±0.014	-2.543±0.115	12.707±0.018	6.861±0.114	2.488±0.039
	VAEAC	NRMSE	0.615±0.000	0.574±0.033	0.896±0.001	0.462±0.002	0.880±0.001
		NLL	1.708±0.005	-2.418±0.006	10.082±0.010	3.452±0.067	0.042±0.002
	AC-Flow	NRMSE	0.603±0.000	0.567±0.050	0.909±0.000	0.478±0.004	0.877±0.001
		NLL	-5.269±0.007	-8.086±0.010	8.197±0.008	0.972±0.022	-0.561±0.003
	AC-Flow BG	NRMSE	0.572±0.000	0.369±0.016	0.861±0.001	0.442±0.001	0.833±0.002
		NLL	-4.841±0.008	-7.593±0.011	6.833±0.006	1.098±0.032	-0.528±0.003
p=0.1	MICE	NRMSE	0.631±0.003	0.518±0.004	0.964±0.004	0.605±0.004	0.911±0.008
	MissForest	NRMSE	0.665±0.002	0.418±0.005	0.985±0.002	0.561±0.003	0.991±0.019
	GAIN	NRMSE	0.749±0.128	0.502±0.070	1.024±0.023	0.615±0.017	1.074±0.038
	AutoEncoder	NRMSE	0.648±0.001	0.761±0.095	0.936±0.001	0.498±0.002	0.887±0.002
		NLL	4.300±0.038	-2.266±0.169	12.851±0.012	7.305±0.043	2.521±0.028
	VAEAC	NRMSE	0.620±0.000	0.558±0.047	0.899±0.000	0.467±0.004	0.881±0.003
		NLL	2.245±0.015	-2.823±0.009	10.389±0.005	4.242±0.071	0.103±0.005
	AC-Flow	NRMSE	0.610±0.000	0.588±0.025	0.908±0.001	0.533±0.005	0.877±0.002
		NLL	-4.225±0.018	-7.568±0.005	7.784±0.006	5.150±0.053	-0.557±0.003
	AC-Flow BG	NRMSE	0.586±0.001	0.384±0.018	0.863±0.001	0.468±0.003	0.836±0.002
		NLL	-3.187±0.017	-7.212±0.008	9.670±0.007	3.577±0.057	-0.510±0.003
p=0.5	MICE	NRMSE	0.628±0.001	0.539±0.005	0.969±0.002	0.615±0.002	0.916±0.005
	MissForest	NRMSE	0.662±0.001	0.436±0.003	0.990±0.002	0.573±0.005	0.990±0.012
	GAIN	NRMSE	0.929±0.123	1.152±0.180	1.143±0.035	0.800±0.042	1.101±0.044
	AutoEncoder	NRMSE	0.739±0.001	0.618±0.056	0.962±0.001	0.567±0.005	0.905±0.002
		NLL	10.078±0.021	0.990±0.097	13.482±0.012	10.775±0.091	2.858±0.047
	VAEAC	NRMSE	0.666±0.001	0.531±0.036	0.915±0.001	0.513±0.004	0.892±0.002
		NLL	9.930±0.029	-1.952±0.023	11.415±0.012	9.051±0.079	0.343±0.004
	AC-Flow	NRMSE	0.667±0.001	0.488±0.030	0.938±0.000	0.614±0.004	0.890±0.000
		NLL	1.508±0.010	-5.405±0.008	10.538±0.006	9.892±0.084	-0.458±0.005
	AC-Flow BG	NRMSE	0.645±0.000	0.421±0.016	0.890±0.000	0.582±0.007	0.843±0.001
		NLL	3.497±0.015	-4.818±0.009	10.975±0.006	10.849±0.105	-0.417±0.005