Maximum Likelihood on the Joint (Data, Condition) Distribution for Solving Ill-Posed Problems with Conditional Flow Models

John S. Hyatt
Computational and Information Sciences Directorate
DEVCOM Army Research Laboratory
Aberdeen Proving Ground, Maryland 21005
john.s.hyatt11.civ@army.mil

Abstract
I describe a trick for training flow models using a prescribed rule as a surrogate for maximum likelihood. The utility of this trick is limited for non-conditional models, but an extension of the approach, applied to maximum likelihood of the joint probability distribution of data and conditioning information, can be used to train sophisticated conditional flow models. Unlike previous approaches, this method is quite simple: it does not require explicit knowledge of the distribution of conditions, auxiliary networks or other specific architecture, or additional loss terms beyond maximum likelihood, and it preserves the correspondence between latent and data spaces. The resulting models have all the properties of non-conditional flow models, are robust to unexpected inputs, and can predict the distribution of solutions conditioned on a given input. They come with guarantees of prediction representativeness and are a natural and powerful way to solve highly uncertain problems. I demonstrate these properties on easily visualized toy problems, then use the method to successfully generate class-conditional images and to reconstruct highly degraded images via super-resolution.

1 Introduction
Consider the problem of reconstructing an image degraded by blurring, low resolution, occlusion, or other effects. If the scale of degradation is small compared to the features in the image, interpolation with some basic prior knowledge about image properties is sufficient to recover the original with a high degree of certainty. On the other hand, if the image is so damaged that small- and medium-scale features are totally lost, the problem is ill-posed and, instead of a single solution, has a highly complex distribution of possible reconstructions consistent with the degraded input, each with a different set of synthesized replacement features.

For aesthetic applications, it may be sufficient only to require one realistic reconstruction. However, applications like uncertainty quantification or risk assessment require the full conditional distribution of possible reconstructions along with a measure of trustworthiness (how well the predicted distribution matches the unknown true distribution).

Flow models are capable of learning and sampling from arbitrary data distributions. Most attempts so far to model conditional probability distributions with flow models have sacrificed some of the properties that make them attractive in the first place, such as interpretability, bijectivity, or a simple loss function. I am aware of one method for conditioning flow models that retains all of these properties; in this paper I develop a new approach, complementary to that one.

Formally, if \( X \) is the space of all possible images in the target domain and \( Y \) the space of degraded images in the same domain, each point \( y \in Y \) defines a conditional distribution, \( p_{X|Y=y}(x) \), of points \( x \in X \). A non-conditional flow model learns to sample from the distribution \( p_X(x) \); my conditional flow model trains on the joint distribution \( p_{X,Y}(x,y) = p_X(x) p_Y(y) \) directly. Specifying \( y \) then gives the conditional distribution \( p_{X|Y=y}(x) \). The resulting conditional model is a direct extension of non-conditional flow models, so it retains all their desirable properties: efficient sampling and inference, a well-behaved and meaningful latent space, and exact reconstruction of encoded latent vectors. It also retains the same weaknesses, such as low model expressivity and difficulty in accurately assigning likelihood to new data, which remain open problems in flow model design.

Because the concept is not tied to the specific applications tested in this paper (class-conditioned generation and super-resolution of images), the model should generalize well to other types of image degradation such as motion blur, noise, missing color channels, and occlusion. Similarly, it should extend to other non-image problems, such as cleaning up noisy audio or radio signals or class- or parameter-conditioned generative modeling of non-image data.

2 Related Work
Generative modeling has been the subject of intensive research for much of the last decade, resulting in a deep and diverse catalog of approaches. The need to guarantee good sampling from an arbitrary data distribution has driven a movement from conventional models towards density models such as normalizing flows.

2.1 Generative Modeling with GANs and VAEs
Historically, the most popular generative machine learning models have been based on Generative Adversarial Networks (GANs, Goodfellow et al. 2014) and Variational Autoencoders (VAEs, Kingma and Welling 2013). Both learn a map \( G : Z \rightarrow X \) from a low-dimensional latent space \( Z \) to a high-dimensional data space \( X \). Since their introductions, GANs and VAEs have benefited from an enormous
amount of research (for surveys, see Jabbar, Li, and Omar 2020 and Wang, She, and Ward 2021). However, GANs and VAEs generally have some undesirable properties.

**Non-likelihood Basis** Ideally, training should maximize the likelihood that \( G \) samples from the true data distribution \( p_X(x) \); or, for a conditional model, from \( p_{X|Y=y}(x) \), given some \( y \). VAEs maximize a lower bound on the likelihood, while the adversarial loss used to train GANs does not incorporate likelihood at all. Flow models maximize likelihood directly, as described in Section 2.2.

\( G(Z) \) and \( X \) Have Disjoint Support The space of generated examples \( G(Z) \) and the data space \( X \) are both low-dimensional manifolds embedded in a much higher-dimensional space (Narayanan and Mitter 2010; Feng, Zhao, and Zha 2021). These manifolds have disjoint support, which causes \( G \) to receive bad gradients during training. Solutions to this problem include revising the objective function (Arjovsky, Chintala, and Bottou 2017; Gulrajani et al. 2017; Chen et al. 2016; Zhao, Song, and Ermon 2019), or adding noise to both \( X \) and \( G(Z) \) (Sønderby et al. 2016).

In contrast, flow models are bijections between \( Z \) and \( X \). Simple noise annealing and data augmentation are enough to ensure overlapping support, as discussed in Section 5.

**\( G \) is Not Invertible** For GANs and VAEs, \( \dim(Z) < \dim(X) \), so \( G^{-1} : X \to Z \) is not well-defined. If \( G \) is a deep neural network, it is also highly nonlinear and therefore not invertible in practice even if \( \dim(Z) = \dim(X) \).

Interpolating from \( z_1 \) to \( z_2 \) in \( Z \) provides a way to interpolate from \( G(z_1) \) to \( G(z_2) \) in \( G(Z) \); if \( z_1 \) and \( z_2 \) correspond to a particular feature axis in the generated data, this can be used to manipulate that data (Radford, Metz, and Chintala 2016). In order to manipulate a real data example \( x \) in this way, it must be first mapped to \( Z \); if \( G \) is not invertible, there is no way to obtain a latent vector corresponding to \( x \). Inverting \( G \) typically means either learning an encoder \( E \approx G^{-1} \) or iteratively updating \( z \) to obtain \( G(z) = x \) for some target \( x \) (for a survey, see Xia et al. 2021).

In contrast, flow models are trivially invertible, providing direct access to latent space. Ideally, the model perfectly maps between \( Z \) and \( X \); then \( z \) and \( x \) are equally good representations because they contain the same information.

**Conditional GANs and VAEs** These combine prior information and random sampling to learn a map \( G : (Z, Y) \to X \). Those capable of realistic and diverse predictions can be very complex (see, for example, Zhu et al., 2017), but their predictions are not guaranteed to be representative of the true data distribution, making them ill-suited for quantitative analysis. Meanwhile, inverting \( G \) is complicated by the fact that \( G^{-1}(x) \) must disentangle \( y \) and \( z \).

The conditional flow model described in this paper is conceptually simple, learning a map \( F : (X, Y) \to (Z, Y) \). Because it maximizes likelihood, its predictions represent the true target distribution to within the capabilities of the chosen model architecture; because it is trivially invertible, the quality of the learned map can be evaluated at low cost using held-out validation data, as shown in Section 6.5.

### 2.2 Likelihood-Based Generative Modeling with Non-conditional Normalizing Flows

Normalizing flow models sidestep most of the problems described in Section 2.1. A non-conditional flow trains on maximum likelihood using change of variables to learn a bijective map \( F : X \to Z \) between data space \( X \) and latent space \( Z \) (Dinh, Krueger, and Bengio 2015; Dinh, SohlDickstein, and Bengio 2016). The latent prior, \( p_Z \), is related to the probability density function of the data, \( p_X \), by

\[
p_X(x) = p_Z(F(x)) | \det J_F(x) | ,
\]

where \( J_F(x) = \partial F(x) / \partial x^T \) is the Jacobian of \( F \). The training objective of a non-conditional flow model \( F \) is

\[
\mathcal{L}^{Flow}(F) = \mathbb{E}_{x \sim p_X^{\text{data}}(x)} \left[ - \log p_X(x) \right] \\
= \mathbb{E}_{x \sim p_X^{\text{data}}(x)} \left[ - \log p_Z(F(x)) - \log | \det J_F(x) | \right].
\]

Once the model is trained, it can then be used to sample \( x \sim p_X(x) \) by evaluating \( F^{-1}(z) \) with \( z \sim p_Z(z) \), or to map new data into latent space as \( F(x) \).

The most popular flow model is Real NVP (Dinh, Sohl-Dickstein, and Bengio 2016), which composes multiple simple neural-network-based affine scaling transformations. More powerful models include Glow (Kingma and Dhariwal 2018), Flow++ (Ho et al. 2019), invertible convolutional flow (Karani et al. 2019), and autoregressive flows (Kingma et al. 2017; Papamakarios, Pavlakou, and Murray 2018). All the experiments in this paper are based on Real NVP, but the conditional model I propose is agnostic to the details of \( F \).

It is important to note that flow models do not solve every problem with generative modeling, as discussed briefly below. These are open research problems.

**Infinitely Many Valid Maps** Eq. 2 does not have a unique global optimum (Nalisnick et al. 2019). Consider the case where both the prior and target distributions are a two-dimensional standard normal Gaussian: \( p_Z(z) = p_X(x) = \mathcal{N}(0, 1) \). The simplest answer is that \( F = \text{the identity map} \), but from the point of view of Eq. 2, infinitely many \( F \) (rotations, reflections, etc.) are equally optimal.

In terms of local minima, the same likelihood can also be achieved for many \( F \), as long as the sum of both terms in Eq. 2 remains constant. A well-optimized \( F \) will output the correct distribution, but without the right inductive biases, the learned map probably operates on undesirable principles (Lan and Dinh 2020).

**Inappropriate Inductive Biases** Normalizing flows have been shown to learn local pixel correlations rather than capture the semantic features of their training data (Kirichenko, Izmailov, and Wilson 2020). As a result, of the many \( F \) with similar \( L^{Flow} \), the model will learn one that assigns low probability to inputs with unusual local correlations, rather than unusual semantic content. \( F \) may therefore not be able to correctly detect inputs with anomalous semantics if it is simply trained according to Eq. 2, which can also lead to sensitivity to adversarial attacks (Pope, Balaji, and Feizi 2019).

Ensuring \( F \) has the correct inductive bias is an open problem in flow model design, and is compounded for conditional flow models, as I show in Section 6.2.
2.3 Modeling Conditional Distributions with Parameterized Flow

In a bijective map, the input and output must have the same dimensionality and contain the same information. In a non-conditional flow, these are \( x \) and \( z \). For a conditional problem, the goal is to sample \( x \sim p_{X|Y=y}(x) \) given \( y \in Y \), so \( x \) is determined by information in both \( y \) and \( z \). What is the correct relationship between \( x, y, \) and \( z? \)

Many conditional flows parameterize a map on the condition as \( F_Y : X \rightarrow Z \), which can be as simple as learning a different non-conditional model for each of a set of discrete conditions \( y \in \{y_1, y_2, \ldots, y_N\} \), and choosing which to apply based on the condition. The alternative is to parameterize the prior as \( p_{Z|Y=y}. \) In both cases, this parameterization is typically at least partially non-invertible.

Prenger et al. (2018), Lu and Huang (2020), Rombach et al. (2020), Ardzzone et al. (2021), and Anantha Padmanabha and Zabaras (2021) parameterize \( F \) by embedding \( y \) into its internal layers, either directly or via an auxiliary network; additional terms must be added to the objective to ensure \( F \) has the correct parameterization. The parameterized map is invertible with respect to \( Z \) and \( X \), but the actual process of parameterizing \( F \) is still non-invertible. Notably, because \( y \) is not part of the flow, it must be embedded separately into every transformation in \( F \). Winkler et al. (2019) follow this approach and additionally parameterize the prior.

Liu et al. (2019) also parameterize the prior with an encoder network \( E \) that non-invertibly combines \( y \) and \( z \) into one latent representation \( z^* = E(z, y) \), with additional loss terms to force \( z^* \) to be similar to \( F(x) \). \( F \) is an invertible map, so no repeated embedding is necessary; however, given a new example \( x \), the model can only obtain \( z^* \), with no way to disentangle \( z \) from \( y \). As a result, this method can only be used to generate data during inference, not map new data back into the original latent space.

Ardizzone et al. (2020) use a Gaussian mixture model as their prior, maximizing the mutual information between the class labels and Gaussian clusters in latent space. The model then classifies inputs based on the region of latent space containing \( F(x) \). Sorrenson et al. (2020) perform a similar parameterization based on independent component analysis.

To my knowledge, the only conditional flow method besides mine that is fully invertible and trained using only maximum likelihood is the recent work of Pumarola et al. (2019); they also optimize on the joint distribution, albeit with a different approach. In an extension of the original flow design, they couple two bijective maps: a non-conditional flow \( F' : Y \rightarrow Z' \) that learns \( p_{Y|X} \), and a conditional model \( F : X \rightarrow (Z, Z') \) that learns \( p_{X|Y=y}. \) Together, they model the joint distribution \( p_{XY}(x, y) = p_{X|Y=y}(x)p_{Y}(y). \) Layers in \( F \) operate on both the output of the previous layer in \( F' \) and that of the corresponding layer in \( F' \). During inference, \( F' \) encodes \( y \) into its own latent representation, while \( z \) is sampled from the latent prior. The two latent vectors are then concatenated, and \( x \sim p_{X|Y=y}(x) \) is generated as \( F^{-1}(z, F'(y)) \). Their approach, where \( p_{Y} \) is directly modeled by \( F' \), is orthogonal to the one described in this paper, where it is implicit in the loss function.

![Figure 1: Comparison between a conventional flow model (left side) and a “backwards-trained” rule-based flow model (right side). Top row: training. Bottom row: inference.](image)

### 3 Flow Models with a Rule-Based Loss

In conventional flow model training, \( p_X \) is unknown and the model must be trained on Eq. 2 to learn the map \( F : X \rightarrow Z \), evaluating its outputs under the latent prior. This makes sense: it is impossible to write in closed form the data distribution \( p_X \) governing a collection of cat photos.

If we know \( p_X \) exactly, however, we can instead train the flow model “backwards” to learn \( F^{-1} : Z \rightarrow X \). The change of variables in this case looks like:

\[
p_Z(z) = p_X(F^{-1}(z)) | \det J_{F^{-1}}(z) |. \tag{3}
\]

However, for the purposes of training a flow model, the exact density function is unnecessary. Consider a simple 2-dimensional dataset where all points lie near the circumference of a circle with radius \( r \). Then \( p_X(x) \) is high-valued when \( x \sim (r \cos(\theta), r \sin(\theta)) \) for all \( \theta \), and low-valued for other \( x \). Knowing nothing but this qualitative shape information, we can substitute \( -\log p_X(x) \rightarrow \lambda |x \cdot x - r^2| \).

It does not matter that this is not a density: during training, \( F \) will be heavily penalized for both \( |F^{-1}(z)| \rightarrow 0 \) and \( |F^{-1}(z)| \rightarrow \infty \), and for the right \( \lambda \), the loss landscape near \( |F^{-1}(z)| = r \) will be “about the same.”

The result of training a simple flow model “backwards” using this rule-based approach is shown in Fig. 1, alongside a conventional model trained on Eq. 2. For the conventional model, \( p_X(x) = p_R(r)p_\theta(\theta), \) with \( p_R(r) = \mathcal{N}(r, \sigma_r^2) \) uniformly distributed angle \( p_\theta(\theta) = U([0, 2\pi]). \) The models learn similar maps (to within topological errors) if \( \lambda \sigma_r \sim 1. \) Other examples are shown in the technical appendix.

More generally, for datasets with tractable density, a surrogate function \( d(x) \) with the right general features can be substituted into Eq. 3. The rule-based analog to Eq. 2 is then

\[
L^{\text{RuleFlow}} = \mathbb{E}_{z \sim p_Z(z)} \left[ - \log p_Z(z) \right] \approx \mathbb{E}_{z \sim p_Z(z)} \left[ \lambda \cdot d(F^{-1}(z)) - \log |\det J_{F^{-1}}(z)| \right], \tag{4}
\]

where \( \lambda \) is a scale factor.

This approach is of limited utility when applied to non-conditional problems. However, as I show in the next section, when applied to conditional problems, it bypasses the need to model the distribution of conditioning information.
4 Modeling the Joint Distribution With Implicitly Parameterized Flow

A fully symmetric conditional map must have conditions associated to both the input and output, such that the total information on both sides of the map is constant. Suppose $Y$ is the condition associated with $X$, and $Y'$ the condition associated with $Z$. If $X$ and $Y$ contain the same information as $Z$ and $Y'$, the bijective map $F : (X, Y) \rightarrow (Z, Y')$ preserves information symmetry and invertibility:

$$
(x, y) = F^{-1}(z, y') = (F^{-1}_X(z, y'), F^{-1}_Y(z, y')) \tag{5a}
$$

$$
(z, y') = F(x, y) = (F_Z(x, y), F_{Y'}(x, y)). \tag{5b}
$$

$F^{-1}_X$ and $F^{-1}_Y$ are the $X$ and $Y$ components of $F^{-1}$, and $F_Z$ and $F_{Y'}$ the $Z$ and $Y'$ components of $F$, respectively. $F$ is otherwise similar to the non-conditional case described in Section 2.2. This formulation has desirable properties: $F$ is fully invertible with respect to both its inputs; the components $(X, Y)$ and $(Z, Y')$ are always separable; and sampling and density estimation each require only one pass.

In general, the relationship between $Y$ and $Y'$ is ambiguous, although the total information in $(X, Y)$ and $(Z, Y')$ must be equal. In the specific case where $Y = Y'$, $F$ defines a fully invertible conditional flow model that can be trained on maximum likelihood and has no need for additional loss terms or non-invertible model components.

4.1 Change of Variables for a Joint Distribution

Analogous to Eq. 1, the change of variables for this map is

$$p_{XY}(x, y) = p_{Y'}(F(x, y)) | \det J_F(x, y)|, \tag{6}
$$

where $p_{XY}(x, y) = p_{XY|Y=y}(x)p_Y(y)$ and $p_{Y'}(z, y') = p_Z(z)p_{Y'}(y')$ are joint probability densities and $J_F(x, y) = \partial F(x, y)/\partial(x, y)^T$ the Jacobian. The latent variable $Z$ is sampled independently from $Y'$; as shown in Section 6, $F$ encodes universal features into $Z$.

Taking the negative log, Eq. 6 can be rewritten as

$$-\log p_{XY}(x, y) = -\log p_Z(F_Z(x, y))
- \log p_{Y'}(F_{Y'}(x, y))
- \log |\det J_F(x, y)|. \tag{7}
$$

The first and third terms are identical to the two terms in Eq. 2, except that they now also depend on $Y$. The second term, containing $p_{Y'}$, does not have a non-conditional counterpart.

$F$ cannot be optimized on Eq. 7 directly without knowing $p_{Y'}$. If $p_{Y'}$ is assumed to have a certain form, this term can be written explicitly: for example, if $p_{Y'}$ is Gaussian with covariance matrix $\Sigma \propto I$, then $- \log p_{Y'}(F_{Y'}(x, y)) = \frac{1}{2\sigma} \parallel F_{Y'}(x, y) - y' \parallel_2^2$, up to a constant term. $p_{Y'}$ can also be modeled with a non-conditional flow, as in Pumarola et al.

Alternatively, it is sufficient to make the weak assumption that $F$ could be optimized—if $p_{Y'}$ were known. If this is true, the only effect of the second term in Eq. 7 is to drive $F_{Y'}(x, y) \rightarrow y'$, regardless of the form of $p_{Y'}$. Then, analogously to Eq. 4, I can replace the second term in Eq. 7 with anything that has the same effect, such as a measure $d(F_{Y'}(x, y), y')$ of the distance between $F_{Y'}(x, y)$ and $y'$. The optimization path will be different, but the final result will be the same as if the model had optimized on Eq. 7, although the true $p_{Y'}$ and $p_{Y'}$ are both still unknown.

Finally, if $Y = Y'$, $p_{Y'}$ and $p_{Y'}$, cancel in Eq. 6, giving a formula for $p_{X|Y=y}(x)$. Conveniently, this can be incorporated into $d$ to give the conditional training objective:

$$
\begin{align*}
L^\text{Flow} &= E_{(x,y) \sim \text{data}}(x,y) \left[ - \log p_{XY}(x, y) \right] \\
&= \mathbb{E}_{(x,y) \sim \text{data}}(x,y) \left[ - \log p_Z(F_Z(x, y)) \right] \\
&\quad + \lambda \cdot d(F_{Y'}(x, y), y') \\
&\quad - \log |\det J_F(x, y)|, \tag{8}
\end{align*}
$$

where $\lambda$ just has to be large enough that the second term dominates the loss at the start of training.\footnote{A similar term is manually added to Eq. 14 of Pumarola et al. (2019) to reinforce the importance of low-frequency cycle consistency between ground truth $x$ and generated $F^{-1}_X(z, y')$.}

$y$ is both an input and an output, so it is implicitly embedded into every layer in $F$, and $F$'s architecture is almost identical to a non-conditional flow, as shown in Fig. 2. As far as I know, no other conditional flow model has this property.

4.2 Training in Two Stages

Training minimizes Eq. 8 using data examples drawn from the joint data distribution, $(x, y) \sim \text{data}(x, y)$. If $\lambda$ is set large enough that the second term dominates early on, training occurs in two distinct stages.

During the first stage, the model trains to both preserve the conditioning input $y$ throughout the map, and satisfy maximum likelihood under $p_{Y'}$. By the time the other two terms begin contributing significantly to the loss, $d(F_{Y'}(x, y), y') \approx 0$. In my experiments, this stage usually finishes before the end of the first epoch.

The second stage occupies the rest of training and proceeds almost identically to the non-conditional case, except that $F$ now depends on both $x$ and $y$. To see this, compare the first and third terms in Eq. 8 to Eq. 2 and write the Jacobian in block matrix form:

$$
J_F(x, y) = \frac{\partial F(x, y)}{\partial(x, y)^T} = \begin{bmatrix}
\frac{\partial F_Z(x, y)}{\partial x^T} & \frac{\partial F_Z(x, y)}{\partial y^T} \\
\frac{\partial F_{Y'}(x, y)}{\partial x^T} & \frac{\partial F_{Y'}(x, y)}{\partial y^T}
\end{bmatrix}.
$$

Figure 2: Schematic of the conditional flow model described in this paper (right), architecturally similar to a non-conditional Real NVP flow model (left) except for the presence of $Y$ in the inputs and outputs. Arrows on the outside of the figure represent squeeze/factor operations.
After the first stage, $F_{Y'}(x, y) \approx y$, so the bottom right term goes to 1. Similarly, $Z$ and $Y'$ are independent, so $F_Z(x, y)$ is independent from $y$ and the top right term goes to 0. Then, 
\[
\det J_F(x, y) = \det \left( \frac{\partial F_Z(x, y)}{\partial x^T} \right),
\]
which closely resembles the Jacobian term in Eq. 2.

4.3 Factoring Out the Condition Distributions

The second stage is equivalent to factoring $p_Y$ out from both sides of Eq. 6 and training on maximum likelihood under the conditional distribution 
\[
p_{X|Y=y}(x) = p_Z(F_Z(x, y)) \left| \det \left( \frac{\partial F_Z(x, y)}{\partial x^T} \right) \right|. 
\]

For discrete $Y$, if all conditional distributions $p_{X|Y=y}^\text{data}(x)$ are well-represented in the training data, a model might bypass the first stage and directly optimize Eq. 9. If $Y$ is continuous and available data consists only of isolated $(x, y)$ pairs, this is not possible. Training must be performed using the joint distribution and Eq. 8 to reach the second stage.

This is distinct from the non-invertible conditional flow models discussed in Section 2.3. Here, $F$ implicitly optimizes $p_Y$ during the first stage, does not require additional loss terms, and $y$ enters invertibly into $F$. Eq. 9 is more similar to the loss function used to train the conditional flow model of Pumarola et al. (2019), with the tradeoff being that instead of effectively two-stage training, their design parameterizes $F$ on $y$ using a two-stage model. At the same time, the implicit condition-dependent parameterization of each layer in my design is learned during training, rather than explicit layer-to-layer coupling as in their two-step process.

5 Implementation Details

My experiments use a modified implementation of Real NVP (Dinh, Sohl-Dickstein, and Bengio 2016), although the concept is equally applicable to other normalizing flows. The code used in this paper is freely available at https://github.com/USArmyResearchLab/ARL-Conditional_Normalizing_Flows. Further experimental details are available in the technical appendix.

Model Architecture  I use a standard normal Gaussian prior, $p_Z(x) \sim \mathcal{N}(0, 1)$, and set $\lambda = 100$ and $d(F_{Y'}(x, y), y) = \|F_{Y'}(x, y) - y\|$ in Eq. 8. $F$ is a Real NVP model built from dense neural networks for the toy problems and multi-scale, ResNeXt-based convolutional neural networks (He et al. 2015; Xie et al. 2016) for the image problems, incorporating identity maps (He et al. 2016) and dilated convolution kernels. I use layer normalization (Ba, Kiros, and Hinton 2016) instead of batch normalization and drop the redundant $L^2$ weight normalization (van Laarhoven 2017). I reshape layer inputs to remove masked elements, rather than replace them with zeros. Squeeze/factor operations pass elements to both the $Z$ and $Y$ components of the output.

Ensuring $F(X', Y)$ Is Supported On $(Z, Y)$  I train $F$ in three steps. First, I pre-condition the model on the latent prior, with $x \sim p_Z(x)$ and $y \sim p_Z(y)$, until it converges. I then anneal in the training and validation data over 100 epochs. Finally, the model trains normally until it converges.

Avoiding Latent Space Sorting  During training, Eq. 8 is evaluated as an average over the batch. For discrete (class) conditions, to guarantee that $F$ does not internally sort each class into a different region of latent space, I segregate the training data by batch: each batch contains examples from only one class. I do not do this for continuous conditions.

Matching Dimensionality Between $Z$ and $Y'$  Images in MNIST and fashion-MNIST have an irregularly shaped, zero-valued background that makes up much of the total image area. $F$ cannot bijectively map the “distribution” of observed values in a background pixel (a delta function) to the latent prior. Similarly, due to the 8-bit compression of the images, foreground pixels are represented by a comb of delta functions corresponding to the discrete intensity values.

The discrete data can be dequantized by replacing each discrete intensity value with a uniform distribution (Uriu, Murray, and Larochelle 2013; Theis, van den Oord, and Bethge 2016). However, the uniform distribution is still bounded, unlike the Gaussian prior, and is therefore not a natural fit for the output of a flow model (Ho et al. 2019). I instead perform a rough unbounded dequantization on the data, $(x, y) \leftarrow (1 - \alpha)(x, y) + \alpha \mathcal{N}(0, 1)$, with $\alpha = 0.02$, regenerating the noise every epoch. This is enough to confuse nearby pixel intensity differences, but not semantic content.

6 Experiments and Discussion

My experiments fall into two sets of overlapping categories: toy vs. image problems, and discrete vs. continuous problems. The toy problems are all 3-dimensional, with $x = (x_1, x_2)$ the Cartesian coordinates of a point and $y$ the corresponding scalar condition, and serve to illustrate model properties. The image problems primarily use fashion-MNIST (Xiao, Rasul, and Vollgraf 2017), with MNIST (Lecun et al. 1998) included for comparison, where the image $x$ and condition $y$ have the same spatial shape.

6.1 Discrete Toy Problem: Interleaved Crescents

Each point $x$ in these experiments is randomly sampled from one of two interleaved half-circles with finite thickness, each associated with a class label $y$. As shown on the left side of Fig. 3, each crescent independently maps to and from the prior. The model performs equally well if the classes overlap or they have different shapes, sizes, and topologies (an example is shown in the technical appendix). It is also robust to out-of-distribution inputs, such as the different $y$ shown on the right side of Fig. 3. Interpolating $y$ between class labels transitions smoothly between them, and the model preserves inputs $F_{Y'}^{-1}(x, y) = y$ (not shown) and qualitative features, even when $y$ is well outside the expected domain.

6.2 Continuous Toy Problem: Randomly Oriented Unit Circle Sectors

The conditional distributions in this case are randomly oriented sectors of the unit circle. To generate each $(x, y)$, I first sample an angle $y$ from the uniform distribution $\mathcal{U}(0, 2\pi)$, and randomly select one point $x$ from the sector of width $1$ radian centered on $y$. I train the model on the full joint distribution; during inference, specifying the angle yields a
Figure 3: Toy discrete problem. Left side: Training (top row) and inference (bottom row) for a flow model $F$ conditioned on class label $y = \pm 1$. Right side: Inference on $F$, with $y \in [-2, 2]$ in increments of 0.5 ($y = \pm 1$ are starred). The predictions degrade gracefully for out-of-distribution $y$.

Figure 4: Inference for toy continuous problem, showing $z = (z_1, z_2)$ and conditional distributions for $y' \in [0, \pi/2, \pi, 3\pi/2]$. In both panels, each point is shaded proportional to $z_1$. The conditional distributions all roughly preserve the spatial orientation of the original $p_Z$.

6.3 Class-Conditional Image Generation

Here, each $x$ is a $28 \times 28$ grayscale image with discrete class label $y$. $y$ is a tensor of the same spatial shape as $x$, where every element $y_{ij} = y$. $(x, y)$ is standardized to mean 0, standard deviation 1 across the entire dataset.

Fig. 5a shows samples generated from two learned conditional distributions with labels “shirt” and “sneaker.” Qualitatively, the images have similar realism and diversity of features as the original dataset. The model also maintains semantic consistency in the latent space across classes (shown in Fig. 5b), and degrades gracefully for out-of-distribution $y$ (Fig. 5c). As with the toy problem, $F$ not only interpolates between true class labels $y_1$ and $y_2$ (the boxed images), it exaggerates class features for $y$ along the same line.

6.4 Super-Resolution of Highly Degraded Images

Here, $F$ reconstructs highly degraded low-resolution images. I spatially downsample the original $28 \times 28$ images using $4 \times 4$ average pooling to obtain $7 \times 7$ inputs. As shown in the first and second columns of Fig. 6, this destroys all but the highest-level features in the original image.

I factor reconstruction into two steps, with two composed models $F = F_1 \circ F_2$. $F_2$ samples from the conditional distribution of possible $14 \times 14$ intermediate images consistent with a given $7 \times 7$ input condition. Then, given a sample from that intermediate distribution, $F_1$ samples from the conditional distribution of possible consistent $28 \times 28$ images.

Both models take their $x$ to be the residual image at that resolution, while $y$ is spatially upscaled to the same size as $x$. This ensures that $x$ and $z$ contain the same information and do not contain $y$, which is not the case if $x$ can simply be downsampled to get $y$. Then $x + y$ gives the predicted image at that resolution. I train the models on the joint distribution $p_{X|Y}^\text{data}(x, y)$, including all 10 classes, without class labels.

Unlike with non-conditional flow models, where $p_{X|Y}^\text{data}(x)$ is available, it is not possible to directly verify that $F$ samples from $p_{X|Y=\hat{y}}(x)$, since (unlike with the toy continuous problem) I do not have access to the distribution of real images that downsample to a given low-resolution $y$. However, Eq. 9 states that this is the case, and the toy experiments in Section 6.2 and the histogram analysis in Section 6.5 provide strong supporting evidence.

Fig. 6 shows the results of this procedure for one example each of fashion-MNIST sneakers and sandals; the model performs equally well on the other 8 classes. The first column represents the ground truth, and the second, the $4 \times 4$-downsampled low-resolution $y$. Note that, after downsampling, almost all features have been lost.

The third column shows 10 predictions from $F_2^{-1}$, which reintroduce intermediate-scale features such as overall shape and pattern. The remaining columns show 10 predictions from $F_1^{-1}$ for each image in column 3, showing variation in smaller details. Each reconstruction can be downsampled to recover its low-resolution condition.

The reconstructions are sometimes ambiguous. For example, although most sandal reconstructions in Fig. 6 resemble the original, some resemble sneakers shaded dark at the front and light at the back, a pattern that occurs occasionally in the training data. This ambiguity is good! The downsampling process destroys most of the information in the original image, so reconstructions should properly reflect that uncertainty. The symmetric and invertible nature of flow models makes it easy to quantitatively verify this as well.

6.5 Quantitative Evaluation of Model Quality

One way to quantify the “trustworthiness” of $F$ is to evaluate the properties of the distributions obtained from validation data. The thickness of the envelope of all distributions (shown for both super-resolution steps in Fig. 7) gives a measure of how well the model map $p_{X|Y=\hat{y}}(x)$ to $p_Z(z)$, and vice versa. If each element in $F_Z(x, y)$ approximates the prior well over $(\hat{x}', \hat{y'})$, the model is sampling from the right data distribution (although $F$ may not be a good map;
see Sections 2.2 and 6.2). To my knowledge, such simple tests have not been performed in the literature on conditional models, and are only very rarely performed on non-conditional models, usually averaged over all latent dimensions (see, for example, Papamakarios et al., 2018, Nalisnick et al., 2019, and Kirichenko et al., 2020). In my experiments, I occasionally find that a small number of element histograms deviate significantly from the prior over \((X, Y)\) (an example is shown in the technical appendix), suggesting that this averaging overlooks some types of error.

Each black line in the figure corresponds to the statistics of a single pixel over the entire validation set, while the white line shows the prior, \(p_Z(z) = \mathcal{N}(0, 1)\). Together, all these histograms fit the prior well, with a fuzzy envelope. Counterintuitively, the super-resolution models have about half the envelope thickness of the class-conditional generative models (not shown), possibly because \(p_{X|Y=y}(x)\) is narrower when \(y\) is a low-resolution image rather than a class label. This also validates the two-step approach to training I take for the super-resolution models in Section 6.4.

Analysis of a flow model’s latent space in the literature is mostly limited to feature algebra. Evaluation of \(F_X^{-1}(z)\) usually focuses on qualitative properties like diversity and realism, or computationally expensive quantitative metrics like the Fréchet Inception Distance (Heusel et al. 2017). The latent space also provides a natural way to independently quantify model quality, for negligible cost, that can also identify potential problem areas during or after training.

7 Conclusion
The conditional flow model described in this paper is a modular extension to standard non-conditional Real NVP, meaning it should be possible to directly apply recent advances in flow model design such as learned masks (Kingma and Dhariwal 2018), improved inductive biases (Kirichenko, Izmailov, and Wilson 2020), or a coupling law that is more expressive (Huang et al. 2018) or incorporates self-attention (Ho et al. 2019). A key part of future work is designing a model tailored to the specific problem at hand.

Because of the “trick” that allows me to avoid modeling \(p_Y(y)\) directly, the condition variable must be the same for both the input and output (i.e., \(Y = Y'\)), meaning this method is unsuitable for problems like domain translation. Otherwise, it provides a simple and inexpensive way to solve ill-posed problems via density modeling. At least in principle, the method provides direct access to the detailed statistics of the predicted distributions; further, the predictions appear robust to out-of-distribution conditioning inputs. Although more work remains to be done, this is a promising, low-cost approach to rigorously solving ill-posed problems.
Acknowledgements

I would like to thank Dr. Michael S. Lee for helpful suggestions. Compute time was provided by the DEVCOM ARL DoD Supercomputing Resource Center.

References

Anantha Padmanabha, G.; and Zabaras, N. 2021. Solving inverse problems using conditional invertible neural networks. Journal of Computational Physics, 433: 110194.

Ardizzone, L.; Kruse, J.; Lüth, C.; Bracher, N.; Rother, C.; and Köthe, U. 2021. Conditional Invertible Neural Networks for Diverse Image-to-Image Translation. In Akata, Z.; Geiger, A.; and Sattler, T., eds., Pattern Recognition, 373–387. Cham: Springer International Publishing.

Ardizzone, L.; Mackowiak, R.; Köthe, U.; and Rother, C. 2020. Exact Information Bottleneck with Invertible Neural Networks: Getting the Best of Discriminative and Generative Modeling. arXiv:2001.06448.

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein GAN. arXiv:1701.07875.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. arXiv:1607.06450.

Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; and Abbeel, P. 2016. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. arXiv:1606.03657.

Dinh, L.; Krueger, D.; and Bengio, Y. 2015. NICE: Non-linear Independent Components Estimation. arXiv:1410.8516.

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

Feng, R.; Zhao, D.; and Zha, Z.-J. 2021. Understanding Noise Injection in GANs. In International Conference on Machine Learning, 3284–3293. PMLR.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Nets. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., Advances in Neural Information Processing Systems 27, 2627–2680. Curran Associates, Inc.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. C. 2017. Improved training of Wasserstein GANs. In Advances in Neural Information Processing Systems, 5767–5777.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Identity mappings in deep residual networks. In European conference on computer vision, 630–645. Springer.

Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; and Hochreiter, S. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc.

Ho, J.; Chen, X.; Srinivas, A.; Duan, Y.; and Abbeel, P. 2019. Flow++: Improving Flow-Based Generative Models with Variational Degquantization and Architecture Design. In Chaudhuri, K.; and Salakhutdinov, R., eds., Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, 2722–2730. PMLR.

Huang, C.; Krueger, D.; Lacoste, A.; and Courville, A. C. 2018. Neural Autoregressive Flows. arXiv:1804.00779.

Jabbar, A.; Li, X.; and Omar, B. 2020. A Survey on Generative Adversarial Networks: Variants, Applications, and Training. arXiv:2006.05132.

Karami, M.; Schuurmans, D.; Sohl-Dickstein, J.; Dinh, L.; and Duckworth, D. 2019. Invertible Convolutional Flow. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d’Alché-Buc, F.; Fox, E.; and Garnett, R., eds., Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc.

Kingma, D. P.; and Dhariwal, P. 2018. Glow: Generative Flow with Invertible 1x1 Convolutions. arXiv:1807.03039.

Kingma, D. P.; Salimans, T.; Jozefowicz, R.; Chen, X.; Sutskever, I.; and Welling, M. 2017. Improving Variational Inference with Inverse Autoregressive Flow. arXiv:1606.04934.

Kirichenko, P.; Izmailov, P.; and Wilson, A. G. 2020. Why normalizing flows fail to detect out-of-distribution data. Advances in Neural Information Processing Systems, 33: 20578–20589.

Lan, C. L.; and Dinh, L. 2020. Perfect density models cannot guarantee anomaly detection. arXiv:2012.03808.

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, R.; Liu, Y.; Gong, X.; Wang, X.; and Li, H. 2019. Conditional Adversarial Generative Flow for Controllable Image Synthesis. arXiv:1904.01782.

Lu, Y.; and Huang, B. 2020. Structured Output Learning with Conditional Generative Flows. Proceedings of the AAAI Conference on Artificial Intelligence, 34(04): 5005–5012.

Nalisnick, E.; Matsukawa, A.; Teh, Y. W.; Gorur, D.; and Lakshminarayanan, B. 2019. Do Deep Generative Models Know What They Don’t Know? arXiv:1810.09136.

Narayanan, H.; and Mitter, S. 2010. Sample Complexity of Testing the Manifold Hypothesis. In Lafferty, J.; Williams, C.; Shawe-Taylor, J.; Zemel, R.; and Culotta, A., eds., Advances in Neural Information Processing Systems, volume 23. Curran Associates, Inc.

Papamakarios, G.; Pavlakou, T.; and Murray, I. 2018. Masked Autoregressive Flow for Density Estimation. arXiv:1705.07057.

Pope, P.; Balaji, Y.; and Feizi, S. 2019. Adversarial Robustness of Flow-Based Generative Models. arXiv:1911.08654.
Prenger, R.; Valle, R.; and Catanzaro, B. 2018. WaveGlow: A Flow-based Generative Network for Speech Synthesis. arXiv:1811.00002.

Pumarola, A.; Popov, S.; Moreno-Noguer, F.; and Ferrari, V. 2019. C-Flow: Conditional Generative Flow Models for Images and 3D Point Clouds. arXiv:1912.07009.

Radford, A.; Metz, L.; and Chintala, S. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434.

Rombach, R.; Esser, P.; and Ommer, B. 2020. Network-to-Network Translation with Conditional Invertible Neural Networks. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., Advances in Neural Information Processing Systems, volume 33, 2784–2797. Curran Associates, Inc.

Sønderby, C. K.; Caballero, J.; Theis, L.; Shi, W.; and Huszar, F. 2016. Amortised MAP Inference for Image Super-resolution. arXiv:1610.04490.

Sorrenson, P.; Rother, C.; and Kothe, U. 2020. Disentanglement by Nonlinear ICA with General Incompressible-flow Networks (GIN). arXiv:2001.04872.

Theis, L.; van den Oord, A.; and Bethge, M. 2016. A note on the evaluation of generative models. In Bengio, Y.; and LeCun, Y., eds., 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings.

Uria, B.; Murray, I.; and Larochelle, H. 2013. RNADE: The real-valued neural autoregressive density-estimator. arXiv:1306.0186.

van Laarhoven, T. 2017. L2 Regularization versus Batch and Weight Normalization. arXiv:1706.05350.

Wang, Z.; She, Q.; and Ward, T. E. 2021. Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy. ACM Comput. Surv., 54(2).

Winkler, C.; Worrall, D. E.; Hoogeboom, E.; and Welling, M. 2019. Learning Likelihoods with Conditional Normalizing Flows. arXiv:1912.00042.

Xia, W.; Zhang, Y.; Yang, Y.; Xue, J.-H.; Zhou, B.; and Yang, M.-H. 2021. GAN Inversion: A Survey. arXiv:2101.05278.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747.

Xie, S.; Girshick, R. B.; Dollár, P.; Tu, Z.; and He, K. 2016. Aggregated Residual Transformations for Deep Neural Networks. arXiv:1611.05431.

Zhao, S.; Song, J.; and Ermon, S. 2019. InfoVAE: Balancing Learning and Inference in Variational Autoencoders. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01): 5885–5892.

Zhu, J.-Y.; Zhang, R.; Pathak, D.; Darrell, T.; Efros, A. A.; Wang, O.; and Shechtman, E. 2017. Toward multimodal image-to-image translation. In Advances in Neural Information Processing Systems, 465–476.
A. Model Architecture

This Appendix describes the architecture of the models I use in my experiments in more detail.

Real NVP Architecture

A non-conditional normalizing flow \( F : X \rightarrow Z \) is defined as the composition of \( N \) simpler bijective transformations, \( F = f_1 \circ f_2 \circ \cdots \circ f_N \). Those transformations map intermediate states, \( f_i : U_i \rightarrow V_i \), where \( U_1 = X, V_i = U_{i+1}, \) and \( V_N = Z \). Each \( u_i \) and \( v_i \) are broken into two blocks:

\[
\begin{align*}
v_{i,1} &= f_{i,1}(u_i) = u_{i,1} \\
v_{i,2} &= f_{i,2}(u_i) = f_{i,2}(u_{i,1}; \phi(u_{i,1}))
\end{align*}
\]  

The first block, \( u_{i,1} \), is unchanged as it passes through \( f_i \), which defines a coupling law, which is an invertible map with respect to its first argument, given the second. \( \phi \) is the coupling function, which can be arbitrarily complex and does not have to be invertible, and is typically defined by one or more neural networks.

The Jacobian of \( F \) is \( |\det J_F| = \prod_i |\det J_{f_i}| \). For an affine coupling law,

\[
\begin{align*}
f_{i,1}(u_i) &= u_{i,1} \\
f_{i,2}(u_i) &= A_i(u_{i,1})u_{i,2} + b_i(u_{i,1}) \\
f_{i,1}^{-1}(v_i) &= v_{i,1} \\
f_{i,2}^{-1}(v_i) &= A_i^{-1}(v_{i,1})(v_{i,2} - b_i(v_{i,1}))
\end{align*}
\]  

where \( A_i \) and \( b_i \) are both functions of \( u_{i,1} = v_{i,1} \). The associated Jacobian and its determinant are then

\[
J_{f_i}(u_i) = \begin{bmatrix}
\frac{\partial f_{i,1}(u_i)}{\partial u_{i,1}} & \frac{\partial f_{i,1}(u_i)}{\partial u_{i,2}} \\
\frac{\partial f_{i,2}(u_i)}{\partial u_{i,1}} & \frac{\partial f_{i,2}(u_i)}{\partial u_{i,2}} \\
& & & 1 \\
\frac{\partial f_{i,2}(u_i)}{\partial u_{i,1}} & \frac{\partial f_{i,2}(u_i)}{\partial u_{i,2}} & A_i
\end{bmatrix},
\]

and \( |\det J_{f_i}(u_i)| = |\det A_i(u_{i,1})| \). Real NVP (Dinh, Sohl-Dickstein, and Bengio 2016) uses scaling affine transformations only \( (A_i \) is diagonal), so \( |\det A_i| = |\prod_j A_{i,j,j}| \).

My conditional flow models have identical architecture, except that they define a map \( F : (X, Y) \rightarrow (Z, Y') \) with \( U_1 = (X, Y), V_N = (Z, Y') \). They are not restricted to Real NVP, but that is the flow model architecture I use for experiments in this paper. \( F \) treats the condition component of the input as an additional channel concatenated to the non-condition (data or prior) component; that is, \( Y \) is concatenated to \( X \) and \( Y' \) is concatenated to \( Z \).

The flow models used for the toy problems are very simple, and do not use compressed masking (described in the next section); each “block” of coupling layers consists of 6 masks, one for each nontrivial combination of the three input dimensions: \( (1, 0, 0), (0, 1, 0), (0, 0, 1), (0, 1, 1), (1, 0, 1), \) and \( (1, 1, 0) \), randomly ordered. All toy experiments in the paper are performed with 4 sets of these coupling blocks (24 coupling layers total), although I observe similar results with as few as 1 block. For the continuous toy problem described in Section 6.2, even with as many as 20 coupling blocks (120 coupling layers total), as expected, the model is still unable to “learn rotations” with a scaling coupling law.

The flow models used for the image problems use coupling blocks with 4 coupling layers each, as there are two complementary checkerboard masks and two complementary channelwise masks. They do use compressed masking (described in the next section). All image experiments in the paper use 4 sets of these coupling blocks (16 coupling layers total), although I observe similar results with between 3 and 6 blocks.

As in the original Real NVP design, for the image problems, I use squeeze/factor operations to factor out elements to pass directly to the output after some layers. Equal numbers of elements are passed to \( Z \) and \( Y' \) per squeeze/factor operation, rather than just \( Z \), as shown in Fig. 2. I observe similar results with or without these squeeze/factor operations (the latter is possible because of the small size of fashion-MNIST and MNIST).
Compressed Masking

In the original Real NVP model, $u_{i,1}$ and $u_{i,2}$ are derived from $u_i$ via masking, using alternating checkerboard and channelwise masks in sets of four. The masked elements are first set to zero. Then, after passing $u_{i,2}$ through $f_{i,2}$, the mask is re-applied (since those elements may be nonzero now) to obtain $v_{i,2}$, and the result added to $v_{i,1} = u_{i,1}$ to obtain $v_i$. The same process is applied in reverse for $f_{i,2}^{-1}$.

Rather than pass extra masked (zeroed) elements through $f_{i,2}$ and re-masking them afterward, I compress my inputs by reshaping $u_i$ to remove the masked elements. With 1-dimensional vector inputs $u_i = [u_{i,1}, u_{i,2}]$, it is possible to define rectangular mask matrices $M_1$ and $M_2$ such that

$$M_1 u_i = u_{i,1}$$

$$M_2 u_i = u_{i,2}$$

$$v_{i,1} M_1^T = [v_{i,1}, 0]$$

$$v_{i,2} M_2^T = [0, v_{i,2}]$$

The relevant components of $u_i$ are reduced in size using $M_1 u_i$, passed into Eqs. A.1a and A.1b, and then projected back into the original space using $v_{i,2} M_2^T$. Analogous transformations can be defined for the checkerboard and channelwise masks used in Real NVP with image-shaped inputs, as shown in Fig. A.1. This reduces the total input size to each neural network by a factor of 2: for checkerboard-masked inputs, both spatial dimensions are halved and the channel depth is doubled; for channelwise-masked inputs, the channel depth is halved.

Neural Network Architecture

For the toy problems, I define $A_i$ and $b_i$ using simple dense neural networks with 12 layers of 64 nodes each. I observe similar results with between 6 and 30 layers, and up to 512 nodes.

For the image problems, I define $A_i$ and $b_i$ using ResNeXt-based convolutional networks (He et al. 2015; Xie et al. 2016) with identity maps (He et al. 2016). Each ResNeXt block contained a branch of $3 	imes 3$ convolutional kernels, separated into a number of sub-branches equal to the cardinality of the network; the experiments in the paper use 64 kernels and a cardinality of 8. Parallel to this branch, there are additional branches of dilated $3 	imes 3$ kernels, with the number of kernels (and cardinality) scaled down by the dilation factor of each dilated branch; the rationale is that fewer kernels should be required to learn long-range pixel correlations in MNIST and Fashion-MNIST. The dilations are chosen as shown in Fig. A.2, with each kernel spanning 2 adjacent pixels in the next-larger dilation. Larger dilations are added following this pattern until the next-larger dilation would be larger than half the spatial size of the input image. This means that for checkerboard-masked coupling layers, there will be one fewer dilation (but because the checkerboard-masked layers are halved spatially during the compression, the total receptive field will be similar).

For the image problems I use Leaky ReLU activations followed by layer normalization (Ba, Kiros, and Hinton 2016), except that the final layers used a tanh activation multiplied by a learned scale factor in $A_i$ and a linear activation in $b_i$. For the toy problems I omit the layer normalization. To keep the first term of Eq. 8 from underflowing, I initialize $A_i$ and $b_i$ with an orthogonal kernel initializer with gain of 0.1.

Logit vs. No Logit

I experimented with training on the logit of intensity for the image problems, as in the original Real NVP paper. I obtain good results, but decided to avoid doing this for the experiments in the paper. The reason is that, both with and without logits, the model sometimes returns very high or low intensities (enough so to be obviously outside the data distribution). Examples of this are shown in Fig. A.3, using a class-conditioned model trained on MNIST digits. The images are scaled to have the same minimum and maximum pixel intensity; some images appear to have unusually light strokes or unusually dark backgrounds are because, without the logit, the model occasionally predicts one or two pixels to be very bright or very dark.
Implementing a logit obscures this: although the logit function ensures that only reasonable predictions are made, it hides the fact that the model itself sometimes predicts obviously out-of-distribution elements. Rather than covering up this problem with the logit, this is evidence that these models require more work before they can be reliably deployed in the real world.

## B. Hardware and Software

The code was written using the TensorFlow library (Abadi et al. 2015) and requires TensorFlow version 2.7 or higher and TensorFlow Probability 0.15 or higher to run. The models are trained on NVIDIA V100 GPUs (32 GB memory) with AMD 7H12 Rome CPUs (Linux OS).

## C. Additional Rule-based Non-conditional Flow Examples

The “backward” rule-based flow training works with other functions whose loss can be written in closed form, as shown in Fig. C.1. The function $\Lambda(\sum_i |x_i| - L)$ (panel a) forces points onto a “diamond” shape with $L^1$ norm $L$. The function $\lambda \prod_i x_i$ (panel b) forces points onto a line with slope $-1$ and intercept 0. The function $\lambda \prod_i x_i$ (panel c) forces points onto a “cross” shape, since the function has a minimum when either component is 0. The function $\lambda |x_1 - x_0^2|$ (panel d) forces points onto a parabola defined by $x_1 = x_0^2$. Larger values of $\lambda$ lead to tighter fits to the target function.

None of these functions are perfect: they do not guarantee that $F$ maps the prior to all points in the desired space, which is impossible for the three unbounded distributions anyway, just that it maps to some of them—particularly those with similar magnitude to the prior. However, for the special case of replacing $p_Y(F_Y(x,y)) \rightarrow d(F_Y(x,y), y')$, rather than mapping the prior, $F$ is mapping the condition input, so the target values are already included in the input and the model simply has to “learn” the identity function.

## D. More Complex Discrete Toy Problem

The conditional flow model works equally well if the classes are dissimilar or overlapping, as shown in Fig. D.1 for a model with four labels, each corresponding to a distinct shape. Topological changes (converting the Gaussian prior to a circle or two distinct blobs) are the main source of error.

## E. Histograms of Data Mapped to $Z$

As mentioned in the text, the histograms of data mapped to latent space (averaged over all the validation data, but not spatially averaged over the image) for the class-conditioned models typically have an envelope about twice as wide for the class-conditioned image generators as the super-resolution models, as shown in Fig. E.1. Additionally, sometimes, the class-conditioned image generators map a small number of pixels to the wrong distribution. Given validation data, they map to something other than the standard normal Gaussian prior, shown by the green lines in the figure. I only observe this occasionally, and only for the class-conditioned models. In the generated data, the affected pixels map the prior to a distribution that skews lighter or darker than the surrounding pixels (and is usually in the background). Spatial averaging of the histograms loses this information, which provides a way to identify a particularly flawed map.

## F. Two-step Super-resolution Model

This appendix contains the algorithms used to train and run inference on the two-step super-resolution model, and more examples from super-resolution on other classes.

### Algorithm F.1 Training the Two-step Super-resolution Model.

| Original dataset contains high-resolution images $x_0$ |
|---|
| $x_0' \leftarrow \text{downsample}(x_0')$ |
| $x_2' \leftarrow \text{downsample}(x_1')$ |
| train $F_2$ on examples $(x_2', y_2)$ |
| $= (x_1' - \text{upsample}(x_2'), \text{upsample}(x_2'))$ |
| train $F_1$ on examples $(x_1', y_1)$ |
| $= (x_0' - \text{upsample}(x_1'), \text{upsample}(x_1'))$ |

### Fashion-MNIST Super-resolution: More Examples

The models $F_1 \circ F_2$ described in Section 6.4 are trained on the full fashion-MNIST dataset, including all 10 classes, without class labels. The paper includes examples from the sneakers and sandals classes; Figs. F.1-F.8 show one example each from the other 8 classes, following the same format. Generally, the model is fairly good at sampling textures and
Figure D.1: Example of a conditional discrete problem where the target distributions overlap (so some points in different conditional distributions are distinguished only by label) and have different shapes and topologies. Left side: target data distributions. Right side: conditional distributions learned by a conditional flow model. Color represents input label on the left and output label on the right (they are identical for all mapped points).

Algorithm F.2 Distribution Inference With the Two-step Super-resolution Model.

Condition input is a single $7 \times 7$ image $y_2$

$N_1 =$ number of $14 \times 14$ reconstructions to sample

$N_2 =$ number of $28 \times 28$ reconstructions to sample

$y_2 \leftarrow$ upsample($y_2$)

for $i$ in $[1, \ldots, N_1]$ do

$z_{2,i} \sim p_{Z_2}(z_2)$

$x_{2,i} \leftarrow F_{2,1}^{-1}(z_{2,i}, y_2)$ \quad \triangleright \text{A point sampled from } p_{x_2|y_2=y_2}(x_2)$

$x'_{2,i} \leftarrow x_{2,i} + y_2$ \quad \triangleright \text{A single possible } 14 \times 14 \text{ reconstruction}

end for

$y_{1,i} \leftarrow$ upsample($x'_{2,i}$)

for $j$ in $[1, \ldots, N_2]$ do

$z_{1,j} \sim p_{Z_1}(z_1)$

$x_{1,ij} \leftarrow F_{1,2}^{-1}(z_{1,j}, y_{1,i})$ \quad \triangleright \text{A point sampled from } p_{x_1|y_1=y_1}(x_1)$

$x'_{0,ij} \leftarrow x_{1,ij} + y_{1,i}$ \quad \triangleright \text{A single possible } 28 \times 28 \text{ reconstruction}

end for

return $\{x'_{0,ij}\}$ \quad \triangleright \text{Samples from the predicted distribution of } 28 \times 28 \text{ reconstructions}

large-scale features such as buckles, collars, or seams. It performs poorly on rarer semantic content such as text. Also note that several classes (particularly the trousers class) are extremely homogeneous over the dataset; most of the variation in reconstructions is packed into the 2% noise added to the training data. Much more so than the continuous toy dataset, there is no reason to think the choice of coupling law (affine scaling transformation) is well-suited to the structure of this dataset. The learned distributions thus likely approximate the true distribution, but the model has probably not learned a good representation of the underlying structure of the problem.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. arXiv:1607.06450.

Dinh, L.; Sohl-Dickstein, J.; and Bengio, S. 2016. Density estimation using Real NVP. arXiv:1605.08803.
Figure E.1: Histograms obtained by feeding validation data through the model in the $XY \rightarrow ZY'$ direction, for class-conditional image generation on (a) MNIST and (b) fashion-MNIST, and for super-resolution on fashion-MNIST, (c) step one and (d) step two. Each pixel is represented by one black line, while the red line shows the prior. In the class-conditional image generation, occasionally the model learns to map a small number of pixels, shown in green, to distributions significantly different from the prior.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Identity mappings in deep residual networks. In European conference on computer vision, 630–645. Springer.

Xie, S.; Girshick, R. B.; Dollár, P.; Tu, Z.; and He, K. 2016. Aggregated Residual Transformations for Deep Neural Networks. arXiv:1611.05431.
Figure F.1: Two-scale super-resolution of an example from the fashion-MNIST T-shirt class. Column 1: original image, $x'_0$. Column 2: $4 \times 4$-downsampled input to the model, $y$. Column 3: 10 different reconstructions of the image at $2 \times 2$-downsampled resolution, $x'_{1,i}$. Columns 4-13: 10 different reconstructions of the image at its original resolution for each of the intermediate reconstructions in column 3, $x'_{0,i,j}$.
Figure F.2: Two-scale super-resolution of an example from the fashion-MNIST trouser class. Column 1: original image, $x'_0$. Column 2: $4 \times 4$-downsampled input to the model, $y$. Column 3: 10 different reconstructions of the image at $2 \times 2$-downsampled resolution, $x'_{1,i}$. Columns 4-13: 10 different reconstructions of the image at its original resolution for each of the intermediate reconstructions in column 3, $x'_{0,ij}$. 
Figure F.3: Two-scale super-resolution of an example from the fashion-MNIST pullover class. Column 1: original image, \( x_0' \). Column 2: 4 × 4-downsampled input to the model, \( y \). Column 3: 10 different reconstructions of the image at 2 × 2-downsampled resolution, \( x_{1,i}' \). Columns 4-13: 10 different reconstructions of the image at its original resolution for each of the intermediate reconstructions in column 3, \( x_{0,i,j}' \).
Figure F.4: Two-scale super-resolution of an example from the fashion-MNIST dress class. Column 1: original image, $x_0'$. Column 2: $4 \times 4$-downsampled input to the model, $y$. Column 3: 10 different reconstructions of the image at $2 \times 2$-downsampled resolution, $x_1',i$. Columns 4-13: 10 different reconstructions of the image at its original resolution for each of the intermediate reconstructions in column 3, $x_{0,i,j}'$. 
Figure F.5: Two-scale super-resolution of an example from the fashion-MNIST coat class. Column 1: original image, $x'_0$. Column 2: $4 \times 4$-downsampled input to the model, $y$. Column 3: 10 different reconstructions of the image at $2 \times 2$-downsampled resolution, $x'_{1,i}$. Columns 4-13: 10 different reconstructions of the image at its original resolution for each of the intermediate reconstructions in column 3, $x'_{0,i,j}$. 
Figure F.6: Two-scale super-resolution of an example from the fashion-MNIST shirt class. Column 1: original image, $x_0'$. Column 2: $4 \times 4$-downsampled input to the model, $y$. Column 3: 10 different reconstructions of the image at $2 \times 2$-downsampled resolution, $x_{1,i}'$. Columns 4-13: 10 different reconstructions of the image at its original resolution for each of the intermediate reconstructions in column 3, $x_{0,ij}'$. 
Figure F.7: Two-scale super-resolution of an example from the fashion-MNIST bag class. Column 1: original image, $x_0'$. Column 2: $4 \times 4$-downsampled input to the model, $y$. Column 3: 10 different reconstructions of the image at $2 \times 2$-downsampled resolution, $x_{1,i}'$. Columns 4-13: 10 different reconstructions of the image at its original resolution for each of the intermediate reconstructions in column 3, $x_{0,ij}'$. 
Figure F.8: Two-scale super-resolution of an example from the fashion-MNIST ankle boot class. Column 1: original image, $x'_0$. Column 2: $4 \times 4$-downsampled input to the model, $y$. Column 3: 10 different reconstructions of the image at $2 \times 2$-downsampled resolution, $x'_{1,i}$. Columns 4-13: 10 different reconstructions of the image at its original resolution for each of the intermediate reconstructions in column 3, $x'_{0,1,i,j}$. 