Generative Model With Dynamic Linear Flow

HUADONG LIAO1, JIAWEI HE2,3, AND KUNXIAN SHU1
1Chongqing Key Laboratory on Big Data for Bio Intelligence, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
2School of Pharmaceutical and Food Sciences, Zhuhai College of Jilin University, Zhuhai 519041, China
3School of Life Sciences, Jilin University, Changchun 130012, China
Corresponding author: Kunxian Shu (shukx@cqupt.edu.cn)

ABSTRACT
Flow-based generative models are a family of exact log-likelihood models with tractable sampling and latent-variable inference, hence conceptually attractive for modeling complex distributions. However, flow-based models are limited by density estimation performance issues as compared to state-of-the-art autoregressive models. Autoregressive models, which also belong to the family of likelihood-based methods, however suffer from limited parallelizability. In this paper, we propose Dynamic Linear Flow (DLF), a new family of invertible transformations with partially autoregressive structure. Our method benefits from the efficient computation of flow-based methods and high density estimation performance of autoregressive methods. We demonstrate that the proposed DLF yields state-of-the-art performance on ImageNet 32 × 32 and 64 × 64 out of all flow-based methods. Additionally, DLF converges significantly faster than previous flow-based methods such as Glow.

INDEX TERMS
Exact likelihood, generative models, invertible transformation.

I. INTRODUCTION
The increasing amount of data, paired with the exponential progress in the capabilities of hardware and relentless efforts for better methods, has tremendously advanced the development in the fields of deep learning, such as image classification [1]–[3] and machine translation [4]–[6]. However, most applications have been greatly limited to situations where high level of supervision is available, as labeling data remains a labor-intensive and cost-inefficient exercise. In the meantime, unlabeled data is generally easier to acquire but its direct utilization is yet a central challenging problem. Deep generative models, an emerging and popular branch of machine learning, aims to address these challenges by modeling the high-dimensional distributions of data p(x) without or requiring minimal supervision and providing useful representations to a number of downstream tasks such as semi-supervised learning [7], synthetic data augmentation [8].

In recent years, the field of generative modeling has advanced significantly, especially in the development and application of generative adversarial networks (GANs) [9] and likelihood-based methods. Likelihood-based generative methods could be further divided into three different categories: variational autoencoders [10], autoregressive models [11]–[14], and flow-based generative methods [15]–[17].

Variational autoencoders have displayed promising parallelizability of training and synthesis, however, it could be technically challenging to optimize the lower bound on the marginal likelihood of the data. Autoregressive models and flow-based generative models both estimate the exact likelihood of the data. However, autoregressive models suffer from the limited parallelizability of synthesis or training, and a lot of effort has been made to overcome this drawback [18]. On the contrary, flow-based generative models are efficient for training and synthesis, but generally yield compromised performance in comparison with autoregressive models in density estimation benchmarks.

In this paper, we focus on the exact likelihood-based methods. In Section II, we first review models of autoregressive methods and flow-based methods. Inspired by their common properties, in Section III, we then propose a new family of invertible transformations with partially autoregressive structure. We illustrate that autoregressive models and flow-based generative models are two extreme forms of our proposed method. In Section VI, Our results show that the proposed method achieves state-of-the-art density estimation performance on ImageNet dataset among flow-based methods and converges significantly faster than Glow model [17], which is the previous state-of-the-art model. Though our method has a partially autoregressive structure, we illustrate that the synthesis of a high-resolution image (i.e., 256 × 256 image) on modern hardware takes less
than one second, which is comparable to most flow-based methods.

II. BACKGROUND

A. FLOW-BASED MODELS

In most flow-based models [15]–[17], the high-dimensional random variable $x$ with complex and unknown true distribution $\sim p^*(x)$ is generally transformed by function $f_0$: $x \rightarrow z$, where $f$ can be any bijective function with explicit inverse $f^{-1} = h$ and is typically composed of a series of invertible transformations $f = f_1 \circ f_2 \circ \cdots \circ f_L$. $z$ has same dimension as $x$ and its probability density $p_0(z)$ is simple and tractable, such as a standard Gaussian distribution. With the rule of change of variable, we then have the marginal log-likelihood of a datapoint and take it as the optimization objective of learning $\theta$:

$$\log p_0(x) = \log p_0(z) + \log |\det(dz/dx)|$$

$$= \log p_0(z) + \sum_{i=1}^{L} \log |\det(dh_i/dh_{i-1})|$$ (1)

where $h_i = f_i(h_{i-1})$ is the hidden output of sequence of transformations, with $h_0 \triangleq x$ and $h_L \triangleq z$.

However, the above formula requires the computation of the absolute value of Jacobian determinant of each intermediate transformation $|\det(dh_i/dh_{i-1})|$, which is generally intractable and therefore, becomes a limitation of the above method. In practice, to overcome this issue, the transformation function $f_i$ is well-designed to let its Jacobian matrix be triangular or diagonal, thus the log-determinant is simply the sum of log-diagonal entries:

$$\log |\det(dh_i/dh_{i-1})| = \sum \log |\text{diag}(dh_i/dh_{i-1})|$$ (2)

In the next part of this section, we will review invertible and tractable transformations reported in previous studies, categorized as fully autoregressive structure and non-autoregressive structure. After that, we will discuss their respective advantages and disadvantages in computational parallelizability and density estimation performance.

B. AUTOREGRESSIVE AND INVERSE AUTOREGRESSIVE TRANSFORMATIONS

Two published studies [19], [20] introduced autoregressive (AR) transformation and Inverse Autoregressive (IAR) transformation, respectively. These methods model a similar invertible and tractable transformation from high-dimensional variable $x$ to $y$:

$$y_i = s_i x_i + \mu_i$$ (3)

where $x_i$ and $y_i$ are the $i$-th element of $x$ and $y$, respectively. The difference between AR and IAR is that $s_i$ and $\mu_i$ are parameterized by different input: $s_i, \mu_i = g(x_{1:i-1})$ in autoregressive transformation and $s_i, \mu_i = g(y_{1:i-1})$ in inverse autoregressive transformation. Here $g$ is an arbitrarily complex function, usually a neural network. The vectorized transformation and its reverse transformation for (inverse) autoregressive transformations could be described as follows:

$$f : y = s \circ x + \mu$$ (4)

$$f^{-1} : x = (y - \mu)/s$$ (5)

where $\circ$ is the Hadamard product or element-wise product, and the addition, division and subtraction are also element-wise operations.

In previous works, AR and IAR have been successfully applied to image generation [20] and speech synthesis [21]. However, as $s_i, \mu_i$ are dependent on previous steps input $x_{1:i-1}$ or output $y_{1:i-1}$, these transformations are inherently sequential in at least one pass of training (IAR) or synthesis (AR), making it difficult to parallelize on modern hardware [18].

C. NON-AUTOREGRESSIVE TRANSFORMATIONS

Non-autoregressive transformations are designed to be parallelizable in both forward and backward pass, with tractable Jacobian determinants and inverses. Here, we describe a number of them:

1) ACTNORM

As one of non-autoregressive transformations, actnorm was proposed to alleviate the training problems encountered in deep models, which is actually a special case of (inverse) autoregressive transformation that the scale $s$ and bias $\mu$ are treated as regular trainable parameters, namely, independent of the input data:

$$y = s \circ x + \mu \quad \text{where} \quad s \text{ and } \mu \text{ are learnable}$$ (6)

It’s worth mentioning that $s$ and $\mu$ are shared between the spatial dimensions of $x$ when the input is 2D images as described in [17].

2) AFFINE/ADDITIVE COUPLING LAYERS

Affine/additive coupling layers [15], [16] split the high-dimensional input $x$ into two parts ($x_1, x_2$) and apply different transformations to each to obtain the output $y = (y_1, y_2)$. The first part is transformed with an identity function thus remains unchanged, and the second part is mapped to a new distribution with an affine transformation:

$$y_1 = x_1$$

$$y_2 = x_2 \circ s + \mu$$ (7)

with $u, s = g(x_1) = g(y_1)$. Same as AR and IAR, $g$ is an arbitrarily complex function, typically a neural network. Note that this transformation can also be rewritten in the same form as (inverse) autoregressive transformations and actnorm method: $y = s' \circ x + \mu'$, where $s' = [1, s]^T$ and $\mu' = [0, \mu]^T$.

These non-autoregressive transformations have the advantage of parallelization, therefore, they are usually faster than the transformations with autoregressive structure. However, previous results have shown that they generally perform much worse in density estimation benchmarks [22].
III. METHOD

In this section, we introduce a new family of transformations, which have the advantages of computational efficiency of non-autoregressive transformations and the high density estimation performance of (inverse) autoregressive transformations.

There are two key observations from the mentioned methods in Section II. First, all methods have a consistent linear estimation performance of (inverse) autoregressive transformations, which have the advantages of computational efficiency. Second, there are two key observations from the mentioned methods: (1) invertible 1-of-K transformation can be considered as the extreme form of piecewise linear function, each of the points having its own weights for affine transformation. (2) In applications, an important concern for dynamic linear transformation is its recursive dependencies in the reverse pass, introduced by that each pair \((s_k, \mu_k)\) depends on previous partition \(x_{k-1}\). We show that this issue could be addressed for two reasons: (1) the recursive dependencies are based on piece and its length of computation is proportional to \(K\), indicating the smaller \(K\) is, the shorter the dependency chain and higher parallelizability we get; (2) in Section VI, our results show that \(K = 2\) is effective enough, which has a similar computational speed compared to non-autoregressive methods, on the contrast, increasing \(K\) leads to no performance gain (Fig. 2).

Similar to the transformations of AR and IAR, we also introduce a variant of dynamic linear transformation. Let \(s_k(\cdot)\) and \(\mu_k(\cdot)\) take the transformed output \(y_{k-1}\) as input instead of \(x_{k-1}\), we then have:

\[
\begin{align*}
\mathbf{f} : \quad y_k &= s(y_{k-1}) \odot x_k + \mu(x_{k-1}) \\
\mathbf{f}^{-1} : \quad x_k &= (y_k - \mu(x_{k-1}))/s(x_{k-1})
\end{align*}
\]

where \(k = 1, 2, \ldots, K\) and initial condition \(y_0 = 1\). We call this variant inverse dynamic linear transformation, which has the same log-determinant as (12).

The Jacobian of the above transformation is triangular with \(s = (s_1, \ldots, s_K)\) as its diagonal elements, thus has a simple log-determinant term:

\[
\log(\det(d\mathbf{y}/d\mathbf{x})) = \sum_{k=1}^{K} \log |s_k|
\]

Note that our proposed transformation can also be rewritten in the following linear form:

\[
\begin{bmatrix}
\mathbf{y} \\
\mathbf{1}
\end{bmatrix} =
\begin{bmatrix}
\mathbf{1} & 0
\end{bmatrix}
\begin{bmatrix}
\mathbf{b} \\
\mathbf{w}
\end{bmatrix}
\begin{bmatrix}
\mathbf{x} \\
\mathbf{1}
\end{bmatrix}
\]

where the variables \(\mathbf{b}' = (\mu(x_0)^T, \ldots, \mu(x_{K-1})^T)\) and \(\mathbf{w} = \text{diag}([s(x_0)^T, \ldots, s(x_{K-1})^T])\), and they are data-dependent.

A. DYNAMIC LINEAR TRANSFORMATION WITH TRIANGULAR JACOBIAN

Let us now consider a high-dimensional variable \(\mathbf{x} \in \mathbb{R}^D\). When splitting it into \(K\) parts along its dimension, we obtain \(\mathbf{x} = (x_1, \ldots, x_K)\), with \(1 \leq K \leq D\). Then we can apply a tractable and bijective function \(y = f(x)\) to it as following:

\[
\begin{align*}
\mathbf{y}_1 &= m(x_1) \\
\mathbf{y}_k &= s(x_{k-1}) \odot x_k + \mu(x_{k-1})
\end{align*}
\]

with \(k = 2, \ldots, K\). Variables \(\mathbf{y}_k, x_k, s_k\) and \(\mu_k\) have the same dimension, and \((s_k, \mu_k) = g_k(x_{k-1})\) are modeled by an arbitrarily complex function (e.g., a neural network) with the previous part of data as input. \(m()\) is simple and bijective with the inverse \(x_k = m^{-1}(y_k)\). An alternative of \(m\) is identity function \(y_1 = x_1\). Then, if combined with \(K = 2\), this transformation turns out to be the case of affine coupling layer, see (7). For more choices, \(m\) could also be invertible \(1 \times 1\) convolution [17], bijective nonlinear function (e.g., tanh, sigmoid, softsign), or combination of them. For simplicity, in the following paper, we choose and discuss the case \(m(x_1) = s_1 \odot x_1 + \mu_1\), where \(s_1\) and \(\mu_1\) are trainable. In other words, \(s_1\) and \(\mu_1\) are modeled by \(g(x_0)\) with that \(x_0\) is any constant, e.g. \(x_0 = 1\). Therefore, formula (9) and its inverse can be rewritten as:

\[
\begin{align*}
\mathbf{f} : \quad y_k &= s(x_{k-1}) \odot x_k + \mu(x_{k-1}) \\
\mathbf{f}^{-1} : \quad x_k &= (y_k - \mu(x_{k-1}))/s(x_{k-1})
\end{align*}
\]

where \(k = 1, 2, \ldots, K\) and initial condition \(x_0 = 1\).
FIGURE 1. Dynamic Linear Flow with multi-scale architecture. At each scale, the input is passed through a squeezing operation to trade the spatial size for the number of channels, followed by $H$ flows of invertible $1 \times 1$ convolution and dynamic linear transformation. The output is split into two halves, one for the next series of flow and another as a part of the final latent variable. The condition $h$ is optional which guides dynamic linear transformation as prior knowledge.

By a factor 2 and transpose them into channels, resulting in $s \times s \times c$ inputs transformed into $\frac{s}{2} \times \frac{s}{2} \times 4c$ tensor. After the squeezing operation, $H$ steps of flows consisting of invertible $1 \times 1$ convolution and dynamic linear transformation are combined into a sequence. Then the output of sequence stacks is factored out half of the dimensions at regular intervals, while all of the another half at different scales are concatenated to obtain the final transformed output. The above operations are iteratively applied for $L$ times.

2) INVERTIBLE $1 \times 1$ CONVOLUTION
To ensure that each dimension can influence every other dimension during the transformation, we apply an invertible $1 \times 1$ convolution layer [17] before each layer of dynamic linear transformation, which is essentially a normal $1 \times 1$ convolution with equal number of input and output channels.

V. RELATED WORK
Exact likelihood-base methods constitute a large family of deep generative models. Autoregressive models [19], [20], [24], one subclass of such methods, use chain rule to decompose the probability distribution $p(x)$ into a product of conditional distributions $p(x_i|x_{<i})$, allowing to model each conditional distribution with flexible structure. This class of models provides efficient density estimation, but is non-parallelizable in training or synthesis and therefore its application is limited. Another subclass, such as affine coupling layer [15]–[17], is typically specified as parallelizable.
invertible transformations, but does not admit satisfactory performance on density estimation.

There is effort that aims to improve the parallelizability of autoregressive models. Multiscale PixelCNN [25] improves sampling speed of autoregressive models by assuming conditional independence of pixels within a sub-patch of image, allowing pixelCNN to sample a patch of pixels in parallel. But its cost is reducing the capacity of the model and hence makes it difficult to capture some statistical patterns. Unlike Multiscale PixelCNN, DLF builds partial autoregressive transformation with as few partitions as possible and stacks it into flow model. This utilizes the high capacity of autoregressive structure and parallelizability of flow models.

On the other side, Flow++ [22] enhances flow-based model by augmenting the elementwise affine transformation of affine coupling layers with logistic mixture probability. This work is an orthogonal approach and can be used together with our method to improve the transformation of each partition \( x_k \) in DLF layers. We omit this logistic mixture transformation from comparisons due to its absence of analytic inverse, which makes it time-consuming to sampling.

### VI. EXPERIMENTS

We evaluate the proposed DLF model on standard image modeling benchmarks such as CIFAR-10 [26], ImageNet [27] among others. We first investigated the impact of number of partitions \( K \) and compared the variants of dynamic linear transformation. With the optimized hyperparameters, we then compare log-likelihood with previous generative models of autoregressive and non-autoregressive families. Lastly, we assess the conditional DLF with class label information and the qualitative aspects of DLF on high-resolution datasets.

In all our experiments, we followed a similar implementation procedure of neural networks \( g_{\theta_k} \) as in Glow, using three convolutional layers but a different activation function for the last convolution layer. More specifically, the first two convolutional layers have \( c \) channels with ReLU activation functions, and \( 3 \times 3 \) and \( 1 \times 1 \) filters, respectively. To control the number of model parameters, \( c \) varies for different number of partitions \( K \) and datasets (Table. 1). The last convolution is \( 3 \times 3 \) with two times of channels of \( x_k \), and its outputs \( o \) are equally split into two parts along the channel dimension, obtained \( \log s_k^t, \mu_k = \text{split}(o) \). For the purpose of training stability, the final \( s_k = \exp(\alpha \tanh(\log s_k^t) + \beta) \), where \( \alpha \) and \( \beta \) are learnable scalar variables. For the conditional DLF, we introduce conditions by \( \log s_k^t, \mu_k = \text{split}(o + Vh) \) in the last layer, where \( V \) is weight matrix for conditioning data. In cases where \( h \) encodes spatial information, the matrix products \( V \) is replaced by a \( 3 \times 3 \) convolution operation. The parameters \( \theta_k \) of neural network are individual between different partitions \( x_k \). Depth \( H \) is always set to 32. See Table. 1 and Appendix A for more details of optimization.

#### 1) EFFECT OF PARTITIONS \( K \) AND MODEL VARIANTS

Choosing a large \( K \) will increase the recursive complexity of the model. Therefore, a small \( K \) is preferred without performance degradation. We tested number of partitions \( K = 2, 4 \) and 6 on CIFAR-10. The number of model parameters is approximately equal to 45M (same size was used on CIFAR-10 in Glow) by controlling channels \( c \), see Table 1. As we can see in Fig. 2, increasing \( K \) is unnecessary and has negative effects on model performance, leading to worse NLL score and slower convergence. On the other hand, we replaced the layers of dynamic linear transformation with its inverse variant when \( K = 2 \), which does not produce significant performance difference. Therefore, we choose \( K = 2 \) and will not evaluate DLF with inverse dynamic linear transformation in the following experiments.

Note that for the case of \( K = 2 \), both the non-inverse and inverse DLF variants start overfitting after 20 epochs.

| Dataset          | Partitions \( K \) | Channels \( c \) | Levels \( L \) | Parameters |
|------------------|--------------------|-----------------|---------------|------------|
| CIFAR-10         | 2                  | 512             | 3             | 44.6M      |
|                  | 4                  | 308             | 3             | 45.5M      |
|                  | 6                  | 246             | 3             | 45.7M      |
| MNIST            | 2                  | 128             | 2             | 1.8M       |
| ImageNet 32×32   | 2                  | 512             | 3             | 44.6M      |
| ImageNet 64×64   | 2                  | 384             | 4             | 50.7M      |
| CelebA HQ 256×256| 2                  | 128             | 6             | 57.4M      |
Table 2. Comparing density estimation performance (bits/dim, lower is better). Results are obtained from 8-bits datasets.

| Family       | Model                  | CIFAR10 | ImageNet 32×32 | ImageNet 64×64 |
|--------------|------------------------|---------|----------------|----------------|
| Non-autoregressive | RealNVP [16]          | 3.49    | 4.28           | 3.98           |
|              | Glow [17]              | 3.35    | 4.09           | 3.81           |
|              | Flow++ [22]            | 3.09    | 3.86           | 3.69           |
|              | DLF (ours)             | 3.44    | 3.85           | 3.57           |
| Autoregressive | Multiscale PixelCNN [25]| -       | 3.95           | 3.70           |
|              | PixelCNN [11]          | 3.00    | 3.86           | 3.63           |
|              | Gated PixelCNN [24]    | 3.03    | 3.83           | 3.57           |
|              | PixelSNAIL [13]        | 2.85    | 3.80           | 3.52           |
|              | SPN [14]               | -       | 3.79           | 3.52           |

And after 50 epochs, the averaged NLL score over epoch on training set reached 3.30 and the loss still keeps decreasing, while the validation NLL increases from 3.51 to 3.55. As mentioned in Section III, dynamic linear transformation is the extreme form of piecewise linear function, learning weights of affine transformation for each input. This indicates that the more powerful the transformation is, the more training data our method is eager for to cover the distribution of whole dataset. Therefore, to avoid overfitting, apart from degrading the capacity of dynamic linear transformation, another approach is to increase the size of training dataset. We will discuss this in greater details in the following sections.

2) DENSITY ESTIMATION
To compare with previous likelihood-based models, we perform density estimation (bits per dimension measure)
FIGURE 8. ImageNet $64 \times 64$ (left) and $32 \times 32$ (right) samples.

on CIFAR10 and ImageNet. In particular, we use the $32 \times 32$ and $64 \times 64$ downsampled version of ImageNet [11]. For all datasets, we follow the same preprocessing (including dataset splitting, dequantization method) as in [17].

On CIFAR10, as discussed earlier, the DLF model with the same size as Glow displayed overfitting. A possible reason is the simplicity and small size of CIFAR10. We tested the assumption by training a same size of model on the relatively complex dataset ImageNet $32 \times 32$. As shown in Table 2, compared to Glow, the improvement is significant by 0.24 bits/dim and overfitting was not observed on ImageNet $32 \times 32$. This encourages us to apply transfer learning to CIFAR10, initializing its parameters with the pre-trained model on ImageNet $32 \times 32$. We found it helpful for the overfitting problem on CIFAR10, obtaining 3.57 bits/dim, while it size is relatively small with 50.7M parameters compared to 112.3M in Glow.

For a further comparison, we ran the following ablation of DLF on ImageNet $32 \times 32$: replacing $y_1 = m(x_1)$ by identity function for case $K = 2$, namely dynamic linear transformation versus affine coupling layer. As shown in Fig. 3, dynamic linear transformation performance significantly better than affine coupling layer. The model with dynamic linear transformation converged within 50 epochs and obtained better NLL score as in Table 2. Though the training for affine coupling layer in this paper is not converged, to our best knowledge, it generally requires hundreds or even thousands of epochs to get converged.

Summarily, the DLF model achieves state-of-the-art density modeling results on ImageNet $32 \times 32$ and $64 \times 64$ among all non-autoregressive models, and is comparable to most autoregressive models.

3) CONDITIONAL DLF
For conditional DLF, we experimented on MNIST [28] and CIFAR10 with class label as prior. The hyperparameters can be found in Table 1. For CIFAR10, only $K = 2$ was tested. For the conditional version, during training, we represent the class label as a 10-dimensional, one-hot encoded vector $h$, and add it to each layer of dynamic linear transformation. On contrary, class label information is not given in the unconditional version. Once converged, we synthesize samples by randomly generating latent variables $z$ from standard Gaussian distribution, and giving one-hot encoded labels to all layers of dynamic linear transformation. As in Fig. 4, the class-conditional samples (sampled after 150 epochs) are controlled by the corresponding label and the quality is better than the unconditional samples (sampled after 200 epochs). This result indicates that DLF correctly learns to control the distribution with class label prior. See Appendix for samples from CIFAR10.

4) SAMPLES AND INTERPOLATION
We present samples randomly generated from the trained DLF model on ImageNet $64 \times 64$ and CelebA HQ $256 \times 256$ [29] in Fig. 5 and 6, both on 8-bit. For CelebA $256 \times 256$ dataset, our model has 57.4M parameters, which is approximately 1/4 of Glow’s, and is trained with 400 epochs. Note that our model has not fully converged on CelebA $256 \times 256$, due to computational constraint.

In Fig. 7, we take pairs of real images from Celeba HQ $256 \times 256$ test set, encode them to obtain the latent
representations, and linearly interpolate between the latents to decode samples. As we can see, the image manifold is smoothly changed.

During sampling, generating a 256 × 256 image at batch size 1 takes about 315ms on a single 1080 Ti GPU, and 1078ms on a single i7-6700k CPU. We believe this sampling speed can be further improved by using inverse dynamic linear transformation, as it has no recursive structure in the reverse computation.

VII. CONCLUSION
We propose a new family of invertible and tractable transformations, coined dynamic linear transformation. Building DLF model with blocks of dynamic linear transformation, we achieved state-of-the-art performance in terms of log-likelihood on ImageNet 32 × 32 and 64 × 64 benchmarks. We also illustrated that our flow-based model can efficiently synthesize high-resolution images.

Flow-based methods optimize exact log-likelihood directly, which is stable and easy for training. With the development of more powerful invertible transformations, we believe flow-based methods will show potential comparable to GANs and give rise to various applications such as speech synthesis and representation for text sequence.

ACKNOWLEDGMENT
(Huadong Liao and Jiawei He contributed equally to this work.)

APPENDIX A
OPTIMIZATION DETAILS
We use the Adam optimizer [30] with $\alpha = 0.005$ and default $\beta_1$ and $\beta_2$. Batch size is 256 for MNIST, 32 for all experiments on CIFAR-10 and ImageNet 32 × 32, 24 for ImageNet 64 × 64, and 8 for CelebA HQ 256 × 256. In practice, the weights of invertible 1 × 1 convolution are possible to become non-invertible thus interrupts the training (especially on CelebA dataset). We found it is caused by the increased and then exploded weights of 1 × 1 convolution during training. Therefore, for CelebA dataset, we use L2 regularization for the weights of invertible 1 × 1 convolution, with $\beta = 1.5 \times 10^{-8}$. During sampling, we use the method proposed in [31] to reduce temperature, which often results in higher-quality images.

APPENDIX B
EXTRA SAMPLES
We present extra samples in Fig. 8 to 9.

REFERENCES
[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.
[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2016, pp. 770–778.
[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jul. 2017, pp. 2261–2269.
[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5998–6008.
[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” Oct. 2018, arXiv:1810.04805. [Online]. Available: https://arxiv.org/abs/1810.04805
[6] A. Radford, J. Wu, R. Child, D. Luan, A. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” OpenAI Blog, vol. 1, p. 87, Feb. 2019.
[7] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, “Semi-supervised learning with deep generative models,” in Proc. Adv. Neural Inf. Process. Syst., 2014, pp. 3581–3589.
